sunsfan的博客

lombok注解介绍

标签： java lombok 注解

2016-12-09 17:49 3077人阅读 评论(0)

分类：

学习经验（14）

目录(?) [+]

lombok注解介绍

lombok注解文档

lombok官网下载

lombok是一个可以帮助我们简化**Java**代码编写的工具类，尤其是简化javabean的编写，即通过采用注解的方式，消除代码中的构造方法，getter/setter等代码，使我们写的类更加简洁，当然，这带来的副作用就是不易阅读...不过，还是能看得懂吧，废话不多说，先看一下lombok支持的一些常见的注解。

1. **@NonNull**
2. **@Cleanup**
3. **@Getter/@Setter**
4. **@ToString**
5. **@EqualsAndHashCode**
6. **@NoArgsConstructor/@RequiredArgsConstructor /@AllArgsConstructor**
7. **@Data**
8. **@Value**
9. **@SneakyThrows**
10. **@Synchronized**
11. **@Log**

@NonNull

常用SQL语句总结 (493)
第一篇博客 (389)
SSH学习记录之Spring（一） (308)

评论排行
J2EE基础知识点总结 (6)
查找算法总结 (0)
记录一下做的几道编程题 (0)
数据库并发控制知识点总结 (0)
Hibernate和Mybatis的区别 (0)
错题记录日记（8.24） (0)
SSH学习日记之Struts（一） (0)
Hibernate知识总结（转载） (0)
SSH学习记录之Spring（一） (0)
MYSQL部分知识点 (0)

推荐文章
* CSDN日报20170828——《4个方法快速打造你的阅读清单》
* CSDN博客模板调查问卷
* 秒杀系统的一点思考
* TCP网络通讯如何解决分包粘包问题
* 技术与技术人员的价值
* GitChat·人工智能 | 除了深度学习，机器翻译还需要啥？

最新评论
J2EE基础知识点总结
lhd963140097 ：谢谢！
J2EE基础知识点总结
zgc187 ：虽热不懂但是挺好的
J2EE基础知识点总结
WineGoGo ：人家叫Java EE好多年了
J2EE基础知识点总结
摆欣安-wakaka ：谢谢博主分享，收获很多
J2EE基础知识点总结
刘雅雯_Viola ：理解力有限~~但还是要赞一个！
J2EE基础知识点总结
king_eagle2015 ：哇塞，这个对java数据类型总结的太详细了

这个注解可以用在成员方法或者构造方法的参数前面，会自动产生一个关于此参数的非空检查，如果参数为空，则抛出一个空指针异常，举个例子来看看：

```
//成员方法参数加上@NonNull注解
public String getName(@NonNull Person p){
    return p.getName();
}
```

实际效果相当于：

```
public String getName(@NonNull Person p){
    if(p==null){
        throw new NullPointerException("person");
    }
    return p.getName();
}
```

用在构造方法的参数上效果类似，就不再举例子了。

@Cleanup

这个注解用在变量前面，可以保证此变量代表的资源会被自动关闭，默认是调用资源的close()方法，如果该资源有其它关闭方法，可使用@Cleanup(“methodName”)来指定要调用的方法，就用输入输出流来举个例子吧：

```
public static void main(String[] args) throws IOException {
     @Cleanup InputStream in = new FileInputStream(args[0]);
     @Cleanup OutputStream out = new FileOutputStream(args[1]);
     byte[] b = new byte[1024];
     while (true) {
       int r = in.read(b);
       if (r == -1) break;
       out.write(b, 0, r);
     }
 }
```

实际效果相当于：

```
public static void main(String[] args) throws IOException {
     InputStream in = new FileInputStream(args[0]);
     try {
       OutputStream out = new FileOutputStream(args[1]);
       try {
         byte[] b = new byte[10000];
         while (true) {
           int r = in.read(b);
           if (r == -1) break;
           out.write(b, 0, r);
         }
       } finally {
         if (out != null) {
           out.close();
         }
       }
     } finally {
       if (in != null) {
         in.close();
       }
     }
}
```

是不是简化了很多。

@Getter/@Setter

这一对注解从名字上就很好理解，用在成员变量前面，相当于为成员变量生成对应的get和set方法，同时还可以为生成的方法指定访问修饰符，当然，默认为public，直接来看下面的简单的例子：

```
1  public class Programmer{
2      @Getter
3      @Setter
4      private String name;
5
6      @Setter(AccessLevel.PROTECTED)
7      private int age;
8
9      @Getter(AccessLevel.PUBLIC)
10     private String language;
11 }
```

实际效果相当于：

```
1  public class Programmer{
2      private String name;
3      private int age;
4      private String language;
5
6      public void setName(String name){
7          this.name = name;
8      }
9
10     public String getName(){
11         return name;
12     }
13
14     protected void setAge(int age){
15         this.age = age;
16     }
17
18     public String getLanguage(){
19         return language;
20     }
21 }
```

这两个注解还可以直接用在类上，可以为此类里的所有**非静态**成员变量生成对应的get和set方法。

@ToString/@EqualsAndHashCode

这两个注解也比较好理解，就是生成toString，equals和hashcode方法，同时后者还会生成一个canEqual方法，用于判断某个对象是否是当前类的实例，生成方法时只会使用类中的**非静态**和**非transient**成员变量，这些都比较好理解，就不举例子了。

当然，这两个注解也可以添加限制条件，例如用@ToString(exclude={“param1”，“param2”})来排除param1和param2两个成员变量，或者用@ToString(of={“param1”，“param2”})来指定使用param1和param2两个成员变量，@EqualsAndHashCode注解也有同样的用法。

@NoArgsConstructor/@RequiredArgsConstr/@AllArgsConstructor

这三个注解都是用在类上的，第一个和第三个都很好理解，就是为该类产生无参的构造方法和包含所有参数的构造方法，第二个注解则使用类中所有带有@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，当然，和前面几个注解一样，成员变量都是**非静态**的，另外，如果类中含有final修饰的成员变量，是无法使用@NoArgsConstructor注解的。

三个注解都可以指定生成的构造方法的访问权限，同时，第二个注解还可以用@RequiredArgsConstructor(staticName=" methodName")的形式生成一个指定名称的静态方法，返回一个调用相应的构造方法产生的对象，下面来看一个生动鲜活的例子：

```
@RequiredArgsConstructor(staticName = "sunsfan")
@AllArgsConstructor(access = AccessLevel.PROTECTED)
@NoArgsConstructor
public class Shape {
    private int x;
    @NonNull
    private double y;
    @NonNull
    private String name;
}
```

实际效果相当于：

```
public class Shape {
    private int x;
    private double y;
    private String name;

    public Shape(){
    }

    protected Shape(int x,double y,String name){
        this.x = x;
        this.y = y;
        this.name = name;
    }

    public Shape(double y,String name){
        this.y = y;
        this.name = name;
    }

    public static Shape sunsfan(double y,String name){
        return new Shape(y,name);
    }
}
```

@Data/@Value

呃!!

@Data注解综合了3,4,5和6里面的@RequiredArgsConstructor注解，其中@RequiredArgsConstructor使用了类中的带有@NonNull注解的或者final修饰的成员变量，它可以使用@Data(staticConstructor=" methodName")来生成一个静态方法，返回一个调用相应的构造方法产生的对象。这个例子就也省略了吧...

@Value注解和@Data类似，区别在于它会把所有成员变量默认定义为private final修饰，并且不会生成set方法。

@SneakyThrows

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，很简单的注解，直接看个例子：

```
public class SneakyThrows implements Runnable {
    @SneakyThrows(UnsupportedEncodingException.class)
    public String utf8ToString(byte[] bytes) {
        return new String(bytes, "UTF-8");
    }

    @SneakyThrows
    public void run() {
        throw new Throwable();
    }
}
```

实际效果相当于：

```
public class SneakyThrows implements Runnable {
    @SneakyThrows(UnsupportedEncodingException.class)
    public String utf8ToString(byte[] bytes) {
        try{
            return new String(bytes, "UTF-8");
        }catch(UnsupportedEncodingException uee){
            throw Lombok.sneakyThrow(uee);
        }
    }

    @SneakyThrows
    public void run() {
        try{
            throw new Throwable();
        }catch(Throwable t){
            throw Lombok.sneakyThrow(t);
        }
    }
}
```

@Synchronized

这个注解用在类方法或者实例方法上，效果和synchronized关键字相同，区别在于锁对象不同，对于类方法和实例方法，synchronized关键字的锁对象分别是类的class对象和this对象，而@Synchronized得锁对象分别是私有静态final对象$LOCK$和私有$final$对象lock，当然，也可以自己指定锁对象，例子也很简单，往下看：

```
public class Synchronized {
    private final Object readLock = new Object();

    @Synchronized
    public static void hello() {
        System.out.println("world");
    }

    @Synchronized
    public int answerToLife() {
        return 42;
    }

    @Synchronized("readLock")
    public void foo() {
        System.out.println("bar");
    }
}
```

实际效果相当于：

```
public class Synchronized {
   private static final Object $LOCK = new Object[0];
   private final Object $lock = new Object[0];
   private final Object readLock = new Object();

   public static void hello() {
     synchronized($LOCK) {
       System.out.println("world");
     }
   }

   public int answerToLife() {
     synchronized($lock) {
       return 42;
     }
   }

   public void foo() {
     synchronized(readLock) {
       System.out.println("bar");
     }
   }
 }
```

@Log

这个注解用在类上，可以省去从日志工厂生成日志对象这一步，直接进行日志记录，具体注解根据日志工具的不同而不同，同时，可以在注解中使用topic来指定生成log对象时的类名。不同的日志注解总结如下(上面是注解，下面是实际作用)：

```
@CommonsLog
private static final org.apache.commons.logging.Log log = org.apache.commons.logging.L
@JBossLog
private static final org.jboss.logging.Logger log = org.jboss.logging.Logger.getLogger
@Log
private static final java.util.logging.Logger log = java.util.logging.Logger.getLogger
@Log4j
private static final org.apache.log4j.Logger log = org.apache.log4j.Logger.getLogger(L
@Log4j2
private static final org.apache.logging.log4j.Logger log = org.apache.logging.log4j.Lo
@Slf4j
private static final org.slf4j.Logger log = org.slf4j.LoggerFactory.getLogger(LogExamp
@XSlf4j
private static final org.slf4j.ext.XLogger log = org.slf4j.ext.XLoggerFactory.getXLogg
```

关于lombok的注解先写到这里，当然，还有其他一些注解需要大家自己去摸索，同时lombok一直在扩展，将来肯定会加入更多的注解元素，拭目以待了。

- 上一篇 TCP/IP面试题
- 下一篇 Storm学习教程

相关文章推荐

- lombok 介绍及基本使用方法
- 【直播】大中型UGC信息网站SEO分享--乔向阳
- lombok.jar
- 【直播】打通Linux脉络 进程、线程和调度--宋宝华
- lombok 简化java代码注解 理解
- 【直播】Java最佳学习路线指导--肖海鹏
- Java注解介绍+代码实现
- 【套餐】C++音视频实战技术套餐--夏曹俊
- lombok注解为java类生成Getter/Setter方法
- 【套餐】0基础拿下HTML5和CSS3--李仁密
- suppressWarnings注解参数介绍
- 【套餐】机器学习之数学基础系列--AI100
- lombok的介绍和使用
- spring注解详细介绍
- lombok介绍及使用
- lombok-1.16.8.jar

查看评论

暂无评论

您还没有登录,请[登录]或[注册]

* 以上用户言论只代表其个人观点，不代表CSDN网站的观点或立场