Lombok 恩恩怨怨

关于Lombok的弃用问题

https://zhuanlan.zhihu.com/p/146659383

01、官网

https://projectlombok.org/

注解文档：https://projectlombok.org/features/all

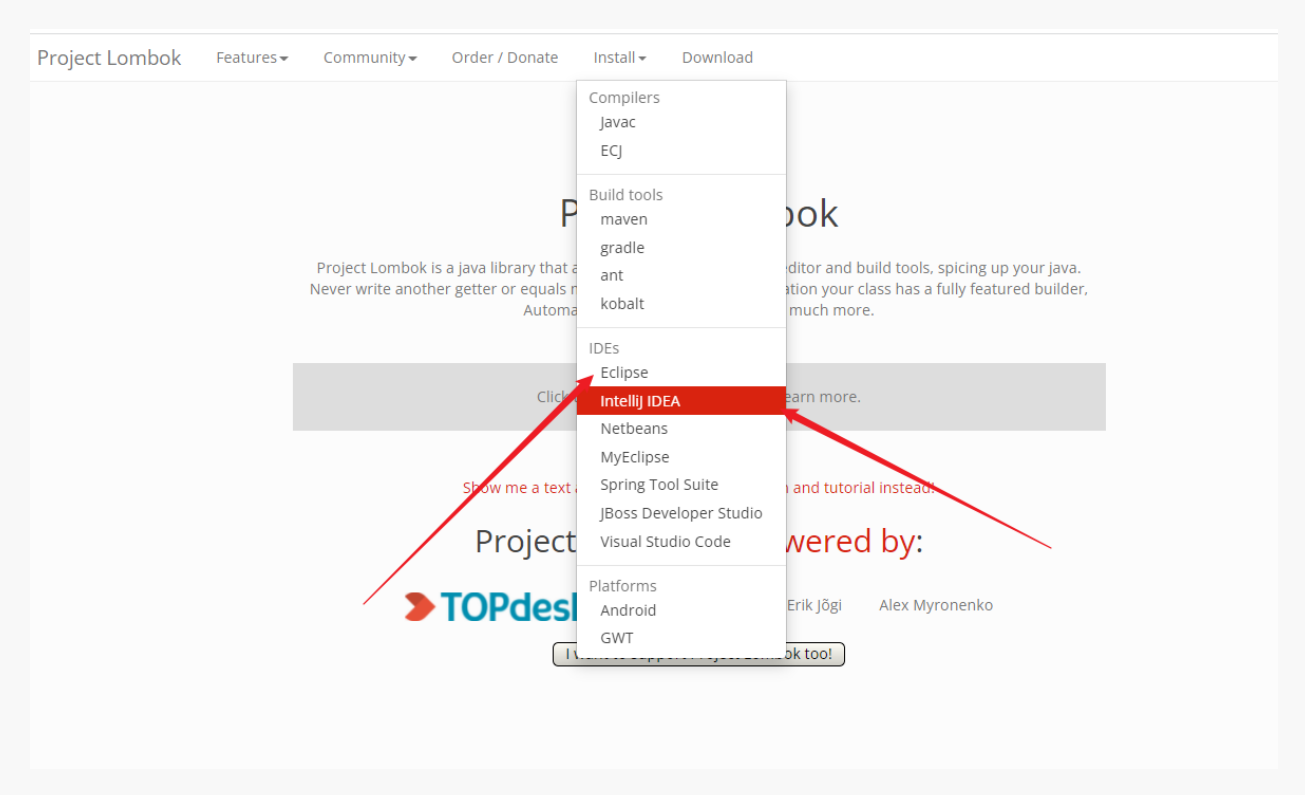
02、说明

Project Lombok 是一个 Java 库，可自动插入您的编辑器并构建工具，为您的 Java 增添趣味。
永远不要再编写另一个 getter 或 equals 方法，通过一个注释，您的类就有一个功能齐全的构建器，自动化您的日志变量。

- 用了lombok可以节约开发时间和效率，不需要在写setter . setter equals tostring等
- 也提供日志的支持 @Slf4j 和 @Log4j2

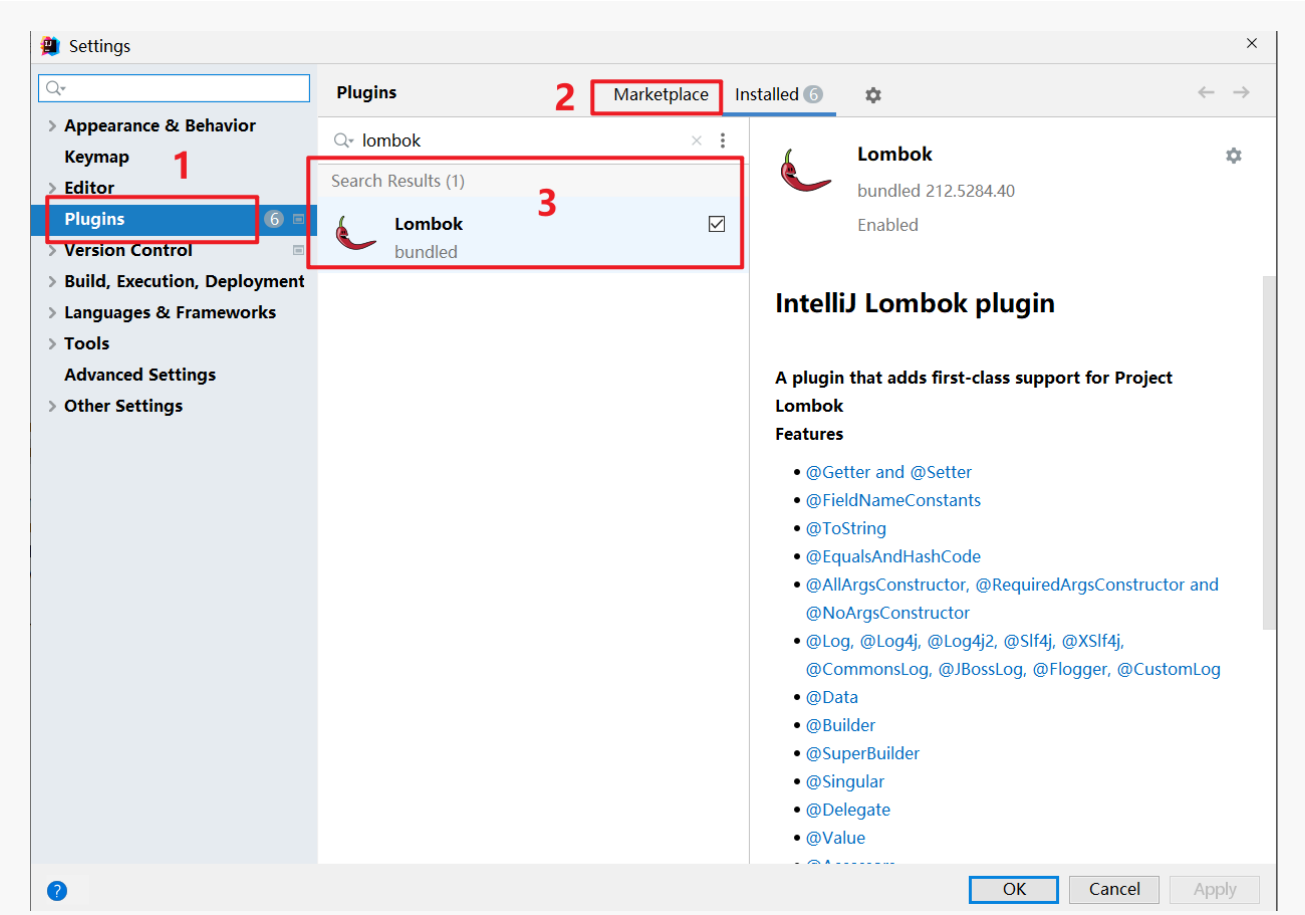
03、使用Lombok的注意事项

在你的idea或者eclipse开发工具中，一定要安装Lombok插件, 否则不会生效。



idea的安装方式：

- Go to `File > Settings > Plugins`
- Click on `Browse repositories...`
- Search for `Lombok Plugin`
- Click on `Install plugin`
- Restart IntelliJ IDEA



04、使用Lombok

lombok：广泛应用于 mybatis-plus 和 spring开源项目都看到它的身影，已经被spring团队拉入的核心依赖中，所以在springboot项目中已经存在lombok。在后续的依赖中不需要在指定版本号。

在项目的pom.xml中依赖lombok

```xml
<!--lombok-->
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <optional>true</optional>
</dependency>
```

04、使用Lombok - 简化bean的定义如下：

@Data注解：它是复合注解包括：

- @ToString
- @EqualsAndHashCode
- @Setter
- @Getter

```java
package com.kuangstudy.bean;

import lombok.Data;

/**
 * Description:
 * Author: yykk Administrator
 * Version: 1.0
 * Create Date Time: 2021/12/14 21:53.
 * Update Date Time:
 *
 * @see
 */
@Data
```

```java
public class User {
    // 主键
    private Integer id;
    // 用户昵称
    private String nickname;
    // 用户密码
    private String password;
    // 用户收集
    private String telephone;
    // 用户的邮箱
    private String email;
    // 用户的头像
    private String avatar;
}

```

编译以后的代码在target目录下，如下：

```java
//
// Source code recreated from a .class file by IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package com.kuangstudy.bean;

public class User {
    private Integer id;
    private String nickname;
    private String password;
    private String telephone;
```

```
13      private String email;
14      private String avatar;
15
16      public User() {
17      }
18
19      public Integer getId() {
20          return this.id;
21      }
22
23      public String getNickname() {
24          return this.nickname;
25      }
26
27      public String getPassword() {
28          return this.password;
29      }
30
31      public String getTelephone() {
32          return this.telephone;
33      }
34
35      public String getEmail() {
36          return this.email;
37      }
38
39      public String getAvatar() {
40          return this.avatar;
41      }
42
43      public void setId(final Integer id) {
```

```
44          this.id = id;
45      }
46
47      public void setNickname(final String
nickname) {
48          this.nickname = nickname;
49      }
50
51      public void setPassword(final String
password) {
52          this.password = password;
53      }
54
55      public void setTelephone(final String
telephone) {
56          this.telephone = telephone;
57      }
58
59      public void setEmail(final String email) {
60          this.email = email;
61      }
62
63      public void setAvatar(final String avatar) {
64          this.avatar = avatar;
65      }
66
67      public boolean equals(final Object o) {
68          if (o == this) {
69              return true;
70          } else if (!(o instanceof User)) {
71              return false;
```

```
72          } else {
73              User other = (User)o;
74              if (!other.canEqual(this)) {
75                  return false;
76              } else {
77                  Object this$id = this.getId();
78                  Object other$id = other.getId();
79                  if (this$id == null) {
80                      if (other$id != null) {
81                          return false;
82                      }
83                  } else if
(!this$id.equals(other$id)) {
84                      return false;
85                  }
86
87                  Object this$nickname =
this.getNickname();
88                  Object other$nickname =
other.getNickname();
89                  if (this$nickname == null) {
90                      if (other$nickname != null)
{
91                          return false;
92                      }
93                  } else if
(!this$nickname.equals(other$nickname)) {
94                      return false;
95                  }
96
```

```
97                  Object this$password =
    this.getPassword();
98                  Object other$password =
    other.getPassword();
99                  if (this$password == null) {
100                     if (other$password != null)
    {
101                         return false;
102                     }
103                 } else if
    (!this$password.equals(other$password)) {
104                     return false;
105                 }
106
107                 label62: {
108                     Object this$telephone =
    this.getTelephone();
109                     Object other$telephone =
    other.getTelephone();
110                     if (this$telephone == null)
    {
111                         if (other$telephone ==
    null) {
112                             break label62;
113                         }
114                     } else if
    (this$telephone.equals(other$telephone)) {
115                         break label62;
116                     }
117
118                     return false;
```

```
119                 }
120 
121                 label55: {
122                     Object this$email =
    this.getEmail();
123                     Object other$email =
    other.getEmail();
124                     if (this$email == null) {
125                         if (other$email == null)
    {
126                             break label55;
127                         }
128                     } else if
    (this$email.equals(other$email)) {
129                         break label55;
130                     }
131 
132                     return false;
133                 }
134 
135                 Object this$avatar =
    this.getAvatar();
136                 Object other$avatar =
    other.getAvatar();
137                 if (this$avatar == null) {
138                     if (other$avatar != null) {
139                         return false;
140                     }
141                 } else if
    (!this$avatar.equals(other$avatar)) {
142                     return false;
```

```
143                 }
144 
145                 return true;
146             }
147         }
148     }
149 
150     protected boolean canEqual(final Object
    other) {
151         return other instanceof User;
152     }
153 
154     public int hashCode() {
155         int PRIME = true;
156         int result = 1;
157         Object $id = this.getId();
158         int result = result * 59 + ($id == null
    ? 43 : $id.hashCode());
159         Object $nickname = this.getNickname();
160         result = result * 59 + ($nickname ==
    null ? 43 : $nickname.hashCode());
161         Object $password = this.getPassword();
162         result = result * 59 + ($password ==
    null ? 43 : $password.hashCode());
163         Object $telephone = this.getTelephone();
164         result = result * 59 + ($telephone ==
    null ? 43 : $telephone.hashCode());
165         Object $email = this.getEmail();
166         result = result * 59 + ($email == null ?
    43 : $email.hashCode());
167         Object $avatar = this.getAvatar();
```

```
168         result = result * 59 + ($avatar == null
    ? 43 : $avatar.hashCode());
169         return result;
170     }
171 
172     public String toString() {
173         return "User(id=" + this.getId() + ",
    nickname=" + this.getNickname() + ", password="
    + this.getPassword() + ", telephone=" +
    this.getTelephone() + ", email=" +
    this.getEmail() + ", avatar=" + this.getAvatar()
    + ")";
174     }
175 }
176 
```

下面代码会怎么样：

```
1 
2 @Data
3 @AllArgsConstructor
4 public class User {}
```

- 上面定义@AllArgsConstructor 生成有参构造函数，无参构造函数就被覆盖。
- 特别在：mybatis-plus , mybatis ,jpa 框中，orm 对象关系映射。因为bean一般是用来和数据库打交道的。存储数据使用，而这些框架的底层都通过反射实例化bean对象，调用bean调用的它的无参构造方法。所以上面代码会报错哦。

- 一句话如果在spring的框架中的任何类上增加了：@AllArgsConstructor 这个注解一定要把无参的也显示的定义出来，否则就引发框架的底层通过反射实例化的时候调不到无参构造函数的问题。

```java
package com.kuangstudy.bean;

import lombok.AllArgsConstructor;
import lombok.Data;
import lombok.NoArgsConstructor;

/**
 * Description:
 * Author: yykk Administrator
 * Version: 1.0
 * Create Date Time: 2021/12/14 21:53.
 * Update Date Time:
 *
 * @see
 */

@Data
@AllArgsConstructor
@NoArgsConstructor
public class User {
    // 主键
    private Integer id;
    // 用户昵称
    private String nickname;
    // 用户密码
    private String password;
```

```
27      // 用户收集
28      private String telephone;
29      // 用户的邮箱
30      private String email;
31      // 用户的头像
32      private String avatar;
33  }
34
```

04、使用Lombok - @Builder

```
1  package com.kuangstudy.bean;
2
3  import lombok.AllArgsConstructor;
4  import lombok.Builder;
5  import lombok.Data;
6  import lombok.NoArgsConstructor;
7
8  /**
9   * Description:
10  * Author: yykk Administrator
11  * Version: 1.0
12  * Create Date Time: 2021/12/14 21:53.
13  * Update Date Time:
14  *
15  * @see
16  */
```

```
17
18 @Data
19 @AllArgsConstructor
20 @NoArgsConstructor
21 @Builder
22 public class User {
23     // 主键
24     private Integer id;
25     // 用户昵称
26     private String nickname;
27     // 用户密码
28     private String password;
29     // 用户收集
30     private String telephone;
31     // 用户的邮箱
32     private String email;
33     // 用户的头像
34     private String avatar;
35 }
36
```

使用

```
1 User user2 = User.builder()
2 .id(1)
3 .avatar("aaa.jpg").build();
```

04-03、Lombok的日志的定义

支持：@Slf4j （logback） 和 @Log4j2 (log4j2)

推荐使用：@Slf4j

springboot的日志支持

```
<!--springboot的web的starter-->
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-
web</artifactId>
</dependency>

```

我们依赖的web中已经包含了日志logging如下：

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-
logging</artifactId>
    <version>2.5.7</version>
    <scope>compile</scope>
</dependency>
```

默认情况：springboot默认日志是：logback

传统的定义日志

在每个类中定义日志

```
1 private final static Logger log =
  LoggerFactory.getLogger(AlipayController.class);
2 private final static Logger log =
  LoggerFactory.getLogger(LoggerController.class);
```

使用

```
1  log.info("-----info--------");
```

Lombok的定义日志

```
1 package com.kuangstudy.web.log;
2
3 import lombok.AllArgsConstructor;
4 import lombok.Synchronized;
5 import lombok.extern.slf4j.Slf4j;
6 import org.slf4j.Logger;
7 import org.slf4j.LoggerFactory;
8 import
  org.springframework.web.bind.annotation.GetMappin
  g;
9 import
  org.springframework.web.bind.annotation.RestContr
  oller;
```

```
10
11 /**
12  * Description:
13  * Author: yykk Administrator
14  * Version: 1.0
15  * Create Date Time: 2021/12/14 15:31.
16  * Update Date Time:
17  *
18  * @see
19  */
20 @RestController
21 @Slf4j
22 public class LoggerController {
23 //    private final static Logger log =
   LoggerFactory.getLogger(LoggerController.class);
24     @GetMapping("/log1")
25     public String log1() {
26         log.info("-----info--------");
27         return "log1";
28     }
29 }
30
```

- 在每个需要增加日志的类上增加注解：@Log4j2 或者 @Slf4j
- 然后在需要打印日志的方法上，通过log对象就去输入日志。

04-04、解决注入springioc注入的对象

在idea中@Autowired注解会出现警告。但是不响应使用，警告idea工具没有读取到实现类。典型的是mybatis的Mapper。所以用@Autowired去注入Mapper出现此警告。解决方案：

- setter方法注入

```
private AlipayProperties alipayProperties;
public void
setAlipayProperties(AlipayProperties
alipayProperties) {
   this.alipayProperties = alipayProperties;
}
```

- 构造函数注入

```
private AlipayProperties alipayProperties;

public AlipayController2(){
}
public AlipayController2(AlipayProperties
alipayProperties ){
    this.alipayProperties = alipayProperties;
}
```

- @Resource

```
@Resource
private AlipayProperties alipayProperties;
```

- Lombok的@RequiredArgsConstructor

```
 */
@RestController
@Slf4j
@RequiredArgsConstructor
public class AlipayController2 {
    private final AlipayProperties
alipayProperties;
}
```

编译以后其实可以发现：

```
//
// Source code recreated from a .class file by
IntelliJ IDEA
// (powered by FernFlower decompiler)
//

package com.kuangstudy.web.properties;

import
com.kuangstudy.web.properties.config.AlipayPro
perties;
import org.slf4j.Logger;
import org.slf4j.LoggerFactory;
import
org.springframework.web.bind.annotation.GetMap
ping;
import
org.springframework.web.bind.annotation.RestCo
ntroller;

```

```java
@RestController
public class AlipayController2 {
    private static final Logger log = LoggerFactory.getLogger(AlipayController2.class);
    private final AlipayProperties alipayProperties;

    @GetMapping({"/alipay2"})
    public String alipay2() {
        log.info("你支付是的:{}", this.alipayProperties);
        return "success";
    }

    public AlipayController2(final AlipayProperties alipayProperties) {
        this.alipayProperties = alipayProperties;
    }
}

```

其实就是一个构造函数注入而已

Lombok的@RequiredArgsConstructor

Lombok的@RequiredArgsConstructor的原理是：构造函数注入