# Lombok

| | |
|---|---|
| Created | @June 6, 2024 1:56 PM |
| Tags | |

Refrence:

> Java - 五分鐘學會 Lombok 用法
> 介紹如何使用 Lombok 幫助我們提升開發效率
> https://kucw.io/blog/2020/3/java-lombok/

Lombok 是一個 Java library，可以透過簡單的注解省略 Java 的 code，像是 setter、getter、logger...等，目的在消除冗長的 code 和提高開發效率

假設你在類上加上了一個 `@Getter` 和 `@Setter` 注解，那你就不用在寫煩人的 getter 和 setter，lombok 會自動幫你產生出來啦！

```java
@Getter
@Setter
public class User {
    private Integer id;
    private String name;
    private String address;
}
```

=

```java
public class User {
    private Integer id;
    private String name;
    private String address;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public String getAddress() {
        return address;
    }

    public void setAddress(String address) {
        this.address = address;
    }
}
```

之所以加個 lombok 的 @Getter 注解就可以幫我們自動生成所有變量的 getter，是因為 lombok 參與了 Java 在 compile 階段生成 `.class` 檔的過程，lombok 會幫我們自動寫一堆 getter，然後塞進 `.class` 檔，所以真正被編譯出來的 User.class 檔案，是包含完整的 getter 的

簡單的說，lombok可以算是一種語法糖，只是在幫我們增進開發效率而已，實際上所產生出來的`.class`檔仍然是完全正常的

## 安裝 Lombok

要在 project 中使用 lombok，除了要在 maven 中加入 lombok dependency，還要安裝 Intellij lombok 插件

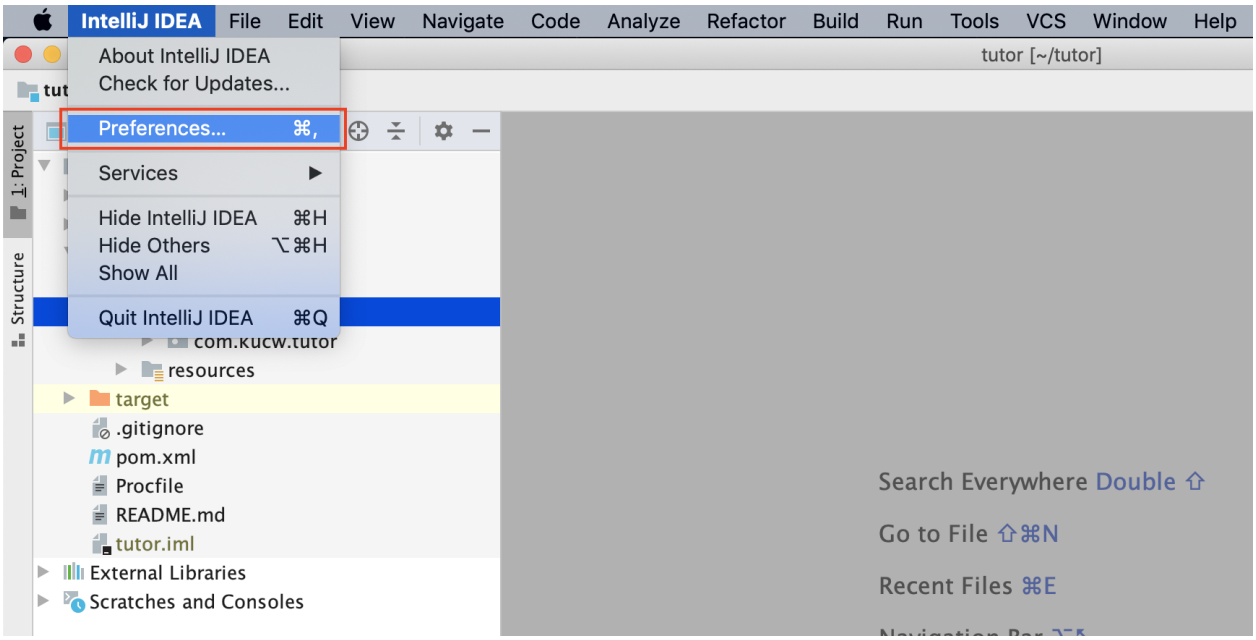
## 1. 加入maven dependency

```
<dependency>
    <groupId>org.projectlombok</groupId>
    <artifactId>lombok</artifactId>
    <version>1.18.18</version>
</dependency>
```

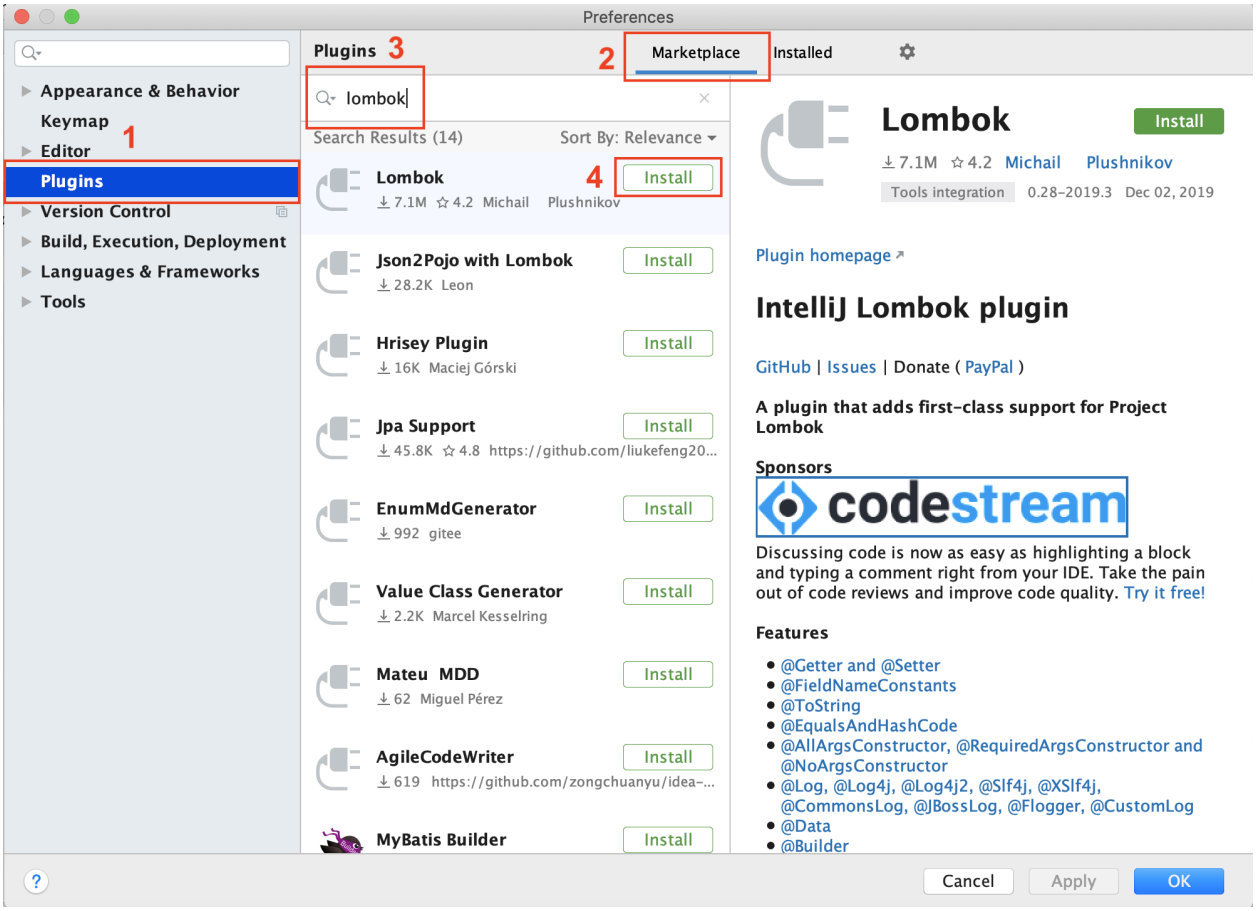
## 2. 在 Intellij 中安裝 lombok 插件

## 我使用的 Intellij 版本是 2019.3.3，可能會因為版本差異導致安裝方式有改變

先點選左上角 Intellij IDEA → Preferences




然後點擊左邊的Plugins，再點擊上面的Marketplace tab，然後就可以在搜尋欄中輸入 `lombok`，並且找到lombok插件並安裝它




## 為什麼需要特地安裝 Intellij lombok 插件？

其實在 maven 加入 lombok dependency 之後，使用 `mvn clean package` 就可以正常 build 過，在 Intellij 中點擊綠色按鈕也可以運行程式

之所以還要特地安裝 Intellij lombok 插件，是因為如果不安裝 lombok 插件的話，Intellij 就會沒辦法自動提示出 lombok 產生的方法，所以就會發生你的 code 一片紅，但是運行卻可以通過的奇妙現象

像是下面這段 code 中，因為對 Intellij 來說，code 裡並不存在 setter，所以沒辦法自動提示 `setId()`、`setName()` 等方法，但是又因為我們在maven中有加入 lombok dependency，所以點擊第 13 行的綠色箭頭運行程式的話，是可以正常運行成功的

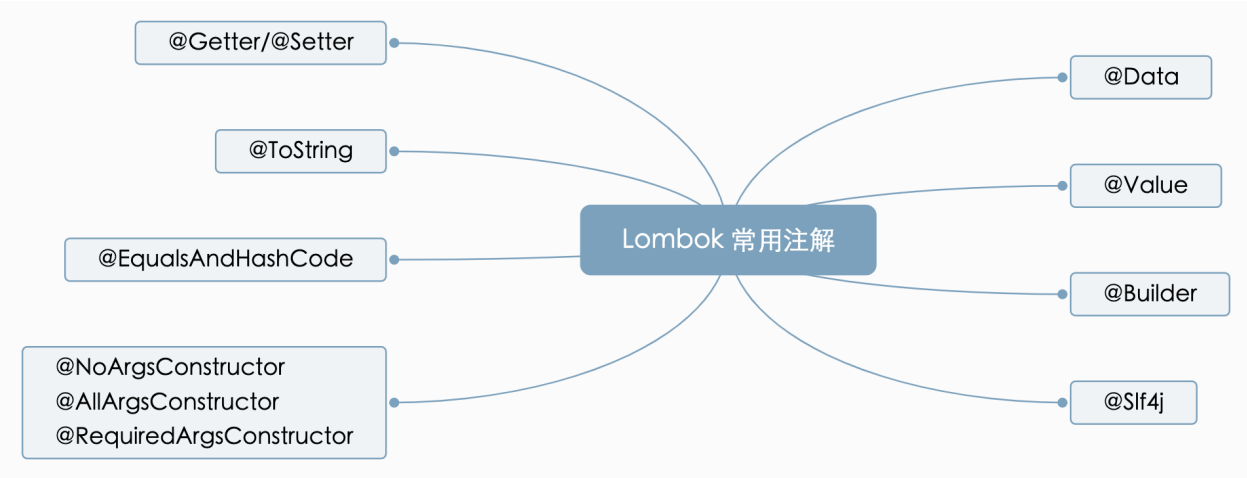
```java
@Getter
@Setter
public class User {
    private Integer id;
    private String name;
}

class Test {
    public static void main(String[] args) {
        User user = new User();
        user.setId(1);
        user.setName("John");
    }
}
```

所以 lombok 算侵入性很高的一個 library，只要團隊中有一個人用 lombok 開發，那麼所有的人都必須得安裝上 lombok 插件，才不會在 Intellij 中一打開 project 時，整片都是痛苦的紅字

## Lombok 用法

lombok 官網提供了許多注解，但是「勁酒雖好，可不能貪杯」，你用了越多 lombok 的進階用法，會讓整個團隊的學習曲線上升，反而會造成反效果，所以在此處只解釋最最常見、並且我認為最必要的注解使用方式，其他的用法就不介紹了



## 1. @Getter/@Setter

自動產生 getter/setter

```java
@Getter
@Setter
public class User {
    private Integer id;
    private String name;
}
```

=

```java
public class User {
    private Integer id;
    private String name;

    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

## 2. @ToString

自動 override `toString()` 方法，會印出所有變量

```java
@ToString
public class User {
    private Integer id;
    private String name;
}
```

=

```java
public class User {
    private Integer id;
    private String name;

    public String toString() {
        return "User(id=" + this.id + ", name=" + this.name + ")";
    }
}
```

## 3. @EqualsAndHashCode

自動生成 `equals(Object other)` 和 `hashcode()` 方法，包括所有非靜態變量和非 transient 的變量

```java
@EqualsAndHashCode
public class User {
    private Integer id;
    private String name;
}
```

=

```java
public class User {
    private Integer id;
    private String name;

    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id) &&
                Objects.equals(name, user.name);
    }

    public int hashCode() {
        return Objects.hash(id, name);
    }
}
```

如果某些變量不想要加進判斷，可以透過 exclude 排除，也可以使用 of 指定某些字段

```java
@EqualsAndHashCode(exclude = "name")
public class User {
    private Integer id;
    private String name;
}
```

=

```java
public class User {
    private Integer id;
    private String name;

    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id);
    }

    public int hashCode() {
        return Objects.hash(id);
    }
}
```

Q：為什麼只有一個整體的 `@EqualsAndHashCode` 注解，而不是分開的兩個 `@Equals` 和 `@HashCode` ？

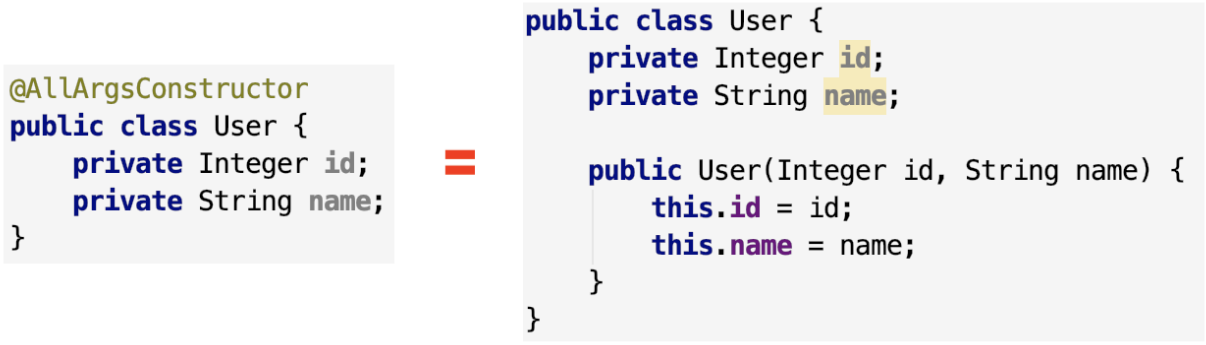
A：在 Java 中有規定，當兩個 object equals 時，他們的 hashcode 一定要相同，反之，當 hashcode 相同時，object 不一定 equals。所以 equals 和 hashcode 要一起 implement，免得發生違反 Java 規定的情形發生

## 4. @NoArgsConstructor, @AllArgsConstructor, @RequiredArgsConstructor

這三個很像，都是在自動生成該類的 constructor，差別只在生成的 constructor 的參數不一樣而已

**@NoArgsConstructor** : 生成一個沒有參數的constructor


**@AllArgsConstructor** : 生成一個包含所有參數的 constructor

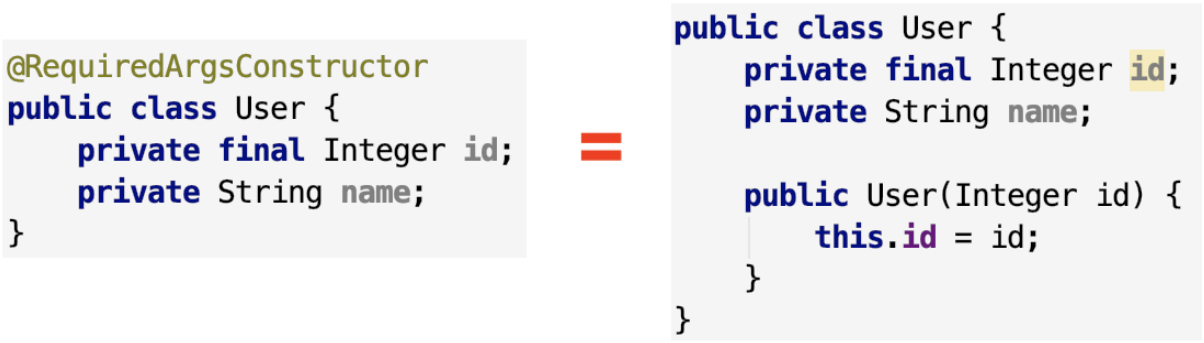



這裡注意一個 Java 的小坑，當我們沒有指定 constructor 時，Java compiler 會幫我們自動生成一個沒有任何參數的 constructor 給該類，但是如果我們自己寫了 constructor 之後，Java 就不會自動幫我們補上那個無參數的 constructor 了

然而很多地方（像是 Spring Data JPA），會需要每個類都一定要有一個無參數的 constructor，所以你在加上 `@AllArgsConstructor` 時，拜託，一定要補上 `@NoArgsConstrcutor`，不然會有各種坑等著你

```
@AllArgsConstructor@NoArgsConstructorpublic class User {
private Integer id;
private String name;
}
```

**@RequiredArgsConstructor** : 生成一個包含 “特定參數” 的 constructor，特定參數指的是那些有加上 final 修飾詞的變量們




補充一下，如果所有的變量都是正常的，都沒有用 final 修飾的話，那就會生成一個沒有參數的 constructor

## 5. @Data

懶人包，只要加了 @Data 這個注解，等於同時加了以下注解

- @Getter/@Setter
- @ToString
- @EqualsAndHashCode
- @RequiredArgsConstructor

```java
@Data
public class User {
    private Integer id;
    private String name;
}
```

=

```java
public class User {
    private Integer id;
    private String name;

    // @RequiredArgsConstructor
    public User() {
    }

    // @Getter/@Setter
    public Integer getId() {
        return id;
    }

    public void setId(Integer id) {
        this.id = id;
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    // @EqualsAndHashCode
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id) &&
                Objects.equals(name, user.name);
    }

    public int hashCode() {
        return Objects.hash(id, name);
    }

    // @ToString
    public String toString() {
        return "User(id=" + this.getId() + ", name=" + this.getName() + ")";
    }
}
```

@Data 是使用頻率最高的lombok注解，通常 @Data 會加在一個值可以被更新的 Object 上，像是日常使用的 DTO 們、或是 JPA 裡的 Entity 們，就很適合加上 @Data 注解，也就是 @Data for mutable class

## 6. @Value

也是懶人包，但是他會把所有的變量都設成 final 的，其他的就跟 @Data 一樣，等於同時加了以下注解

- @Getter (注意沒有setter)
- @ToString

- @EqualsAndHashCode
- @RequiredArgsConstructor

```java
@Value
public class User {
    private Integer id;
    private String name;
}
```

=

```java
public class User {
    private final Integer id;
    private final String name;

    // @RequiredArgsConstructor
    public User(final Integer id, final String name) {
        this.id = id;
        this.name = name;
    }

    // @Getter
    public Integer getId() {
        return id;
    }

    public String getName() {
        return name;
    }

    // @EqualsAndHashCode
    public boolean equals(Object o) {
        if (this == o) return true;
        if (o == null || getClass() != o.getClass()) return false;
        User user = (User) o;
        return Objects.equals(id, user.id) &&
                Objects.equals(name, user.name);
    }

    public int hashCode() {
        return Objects.hash(id, name);
    }

    // @ToString
    public String toString() {
        return "User(id=" + this.getId() + ", name=" + this.getName() + ")";
    }
}
```

上面那個 @Data 適合用在 POJO 或 DTO 上，而這個 @Value 注解，則是適合加在值不希望被改變的類上，像是某個類的值當創建後就不希望被更改，只希望我們讀它而已，就適合加上 @Value 注解，也就是 @Value for immutable class

另外注意一下，此 lombok 的注解 @Value 和另一個 Spring 的注解 @Value 撞名，在 import 時不要 import 錯了

## 7. @Builder

自動生成流式 set 值寫法，從此之後再也不用寫一堆 setter 了

```java
@Builder
public class User {
    private Integer id;
    private String name;
}
```

```java
public static void main(String[] args) {
    User user = User.builder().id(1).name("John").build();
}
```

=

```java
public class User {
    private Integer id;
    private String name;

    public void setId(Integer id) {
        this.id = id;
    }

    public void setName(String name) {
        this.name = name;
    }
}
```

```java
public static void main(String[] args) {
    User user = new User();
    user.setId(1);
    user.setName("John");
}
```

注意，雖然只要加上 @Builder 注解，我們就能夠用流式寫法快速設定 Object 的值，但是 setter 還是必須要寫不能省略的，因為 Spring 或是其他框架有很多地方都會用到 Object 的 getter/setter 對他們取值/賦值

所以通常是 @Data 和 @Builder 會一起用在同個類上，既方便我們流式寫 code，也方便框架做事

```java
@Data@Builderpublic class User {

private Integer id;

private String name;
}
```

## 8. @Slf4j

自動生成該類的 log 靜態常量，要打日誌就可以直接打，不用再手動 new log 靜態常量了

```java
@Slf4j
public class User {
    public static void main(String[] args) {
        log.info("hello");
    }
}
```

=

```java
public class User {

    private static final Logger log = LoggerFactory.getLogger(User.class);

    public static void main(String[] args) {
        log.info("hello");
    }
}
```

除了 @Slf4j 之外，lombok 也提供其他日誌框架的變種注解可以用，像是 @Log、@Log4j...等，他們都是幫我們創建一個靜態常量 log，只是使用的 library 不一樣而已

```java
@Log //對應的log語句如下 private static final java.util.logging.Logger log = java.util.logging.Logger.getLogger(LogExample.class.getName());
```

```
@Log4j    //對應的log語句如下 private    static    final  org.apache.log4j.Logger log  =
org.apache.log4j.Logger.getLogger(LogExample.class);
```

SpringBoot默認支持的就是 slf4j + logback 的日誌框架，所以也不用再多做啥設定，直接就可以用在 SpringBoot project上，log 系列注解最常用的就是 @Slf4j