## 下载并引用

### maven项目添加依赖

**[html]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **<dependency>**
2. **<groupId>**org.projectlombok**</groupId>**
3. **<artifactId>**lombok**</artifactId>**
4. **<version>**1.16.16**</version>**
5. **</dependency>**

或者到官网下载jar包 [https://projectlombok.org/download](https://projectlombok.org/download" \t "http://blog.csdn.net/v2sking/article/details/_blank)

## **安装ide插件**

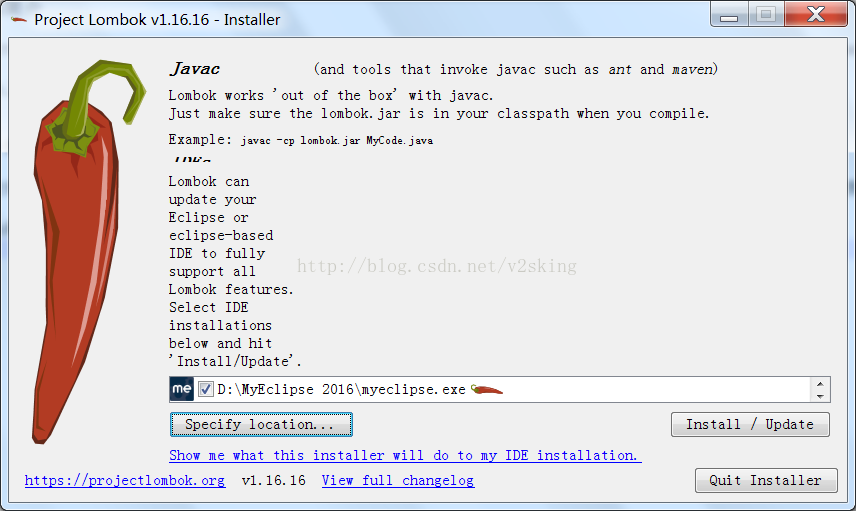
### **myeclipse/eclipse**

下载完成后 命令行运行

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. java -jar lombok-1.16.16.jar

弹出安装界面：



specify location 选择myeclipse安装目录，eclipse同理。

点击 install/update 安装完成。

或者将jar包放入myeclipse 根目录下

myeclipse.ini文件末尾添加：

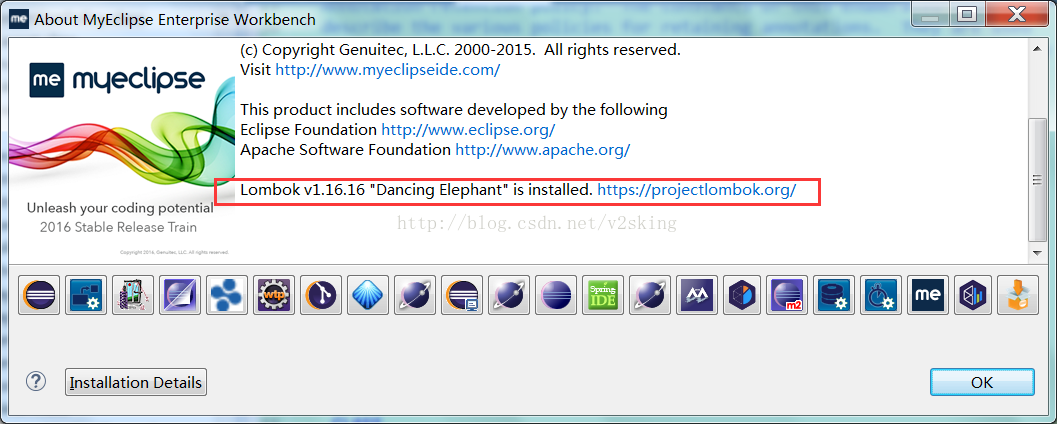
**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. -javaagent:lombok-1.16.16.jar

重启myeclipse即可。

安装完毕后

打开myeclipse about 可以看到



证明插件安装完成

### **IntelliJ IDEA**

* 定位到 File > Settings > Plugins
* 点击 Browse repositories…
* 搜索 Lombok Plugin
* 点击 Install plugin
* 重启 IDEA

## **Lombok注解详解**

### **全局配置文件**

我们可以从项目根目录下新建一个lombok.config（当然目录不是固定的，lombok会搜索所有lombok.config文件）

在这个文件加入一行

config.stopBubbling = true

表示该文件目录为根目录，lombok将从该目录下开始搜索。

每个子目录都可以配置lombok.config 作用范围只在该目录下，并且覆盖父目录的配置。

Lombok通常为所有生成的节点生成注释,添加@javax.annotation.Generated 。

可以用:

lombok.addJavaxGeneratedAnnotation = false 设置取消

下面看下lombok提供了哪些有趣的注解。

### **1.@val @var**

使用Lombok ，java也能够像javascript一样使用弱类型定义变量了

val注解变量申明是final类型 var注解变量是非final类型

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** java.util.ArrayList;
2. **import** java.util.HashMap;
3. **import** lombok.val;
5. **public** **class** ValExample {
6. **public** String example() {
7. val example = **new** ArrayList<String>();
8. example.add("Hello, World!");
9. val foo = example.get(0);
10. **return** foo.toLowerCase();
11. }
13. **public** **void** example2() {
14. val map = **new** HashMap<Integer, String>();
15. map.put(0, "zero");
16. map.put(5, "five");
17. **for** (val entry : map.entrySet()) {
18. System.out.printf("%d: %s\n", entry.getKey(), entry.getValue());
19. }
20. }
21. }

翻译后

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. <span style="font-weight:normal;">**import** java.util.ArrayList;
2. **import** java.util.HashMap;
3. **import** java.util.Map;
5. **public** **class** ValExample {
6. **public** String example() {
7. **final** ArrayList<String> example = **new** ArrayList<String>();
8. example.add("Hello, World!");
9. **final** String foo = example.get(0);
10. **return** foo.toLowerCase();
11. }
13. **public** **void** example2() {
14. **final** HashMap<Integer, String> map = **new** HashMap<Integer, String>();
15. map.put(0, "zero");
16. map.put(5, "five");
17. **for** (**final** Map.Entry<Integer, String> entry : map.entrySet()) {
18. System.out.printf("%d: %s\n", entry.getKey(), entry.getValue());
19. }
20. }
21. }</span>

2.@NonNull

### 在方法或构造函数的参数上使用@NonNull，lombok将生成一个空值检查语句。

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. <span style="font-weight:normal;"> **import** lombok.NonNull;
3. **public** **class** NonNullExample **extends** Something {
4. **private** String name;
6. **public** NonNullExample(@NonNull Person person) {
7. **super**("Hello");
8. **this**.name = person.getName();
9. }
10. }</span>

翻译后

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. <span style="font-weight:normal;">**import** lombok.NonNull;
3. **public** **class** NonNullExample **extends** Something {
4. **private** String name;
6. **public** NonNullExample(@NonNull Person person) {
7. **super**("Hello");
8. **if** (person == **null**) {
9. **throw** **new** NullPointerException("person");
10. }
11. **this**.name = person.getName();
12. }
13. }</span>

3.@Cleanup

### 使用该注解能够自动释放io资源

### 

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. <span style="font-weight:normal;"> **import** lombok.Cleanup;
2. **import** java.io.\*;
4. **public** **class** CleanupExample {
5. **public** **static** **void** main(String[] args) **throws** IOException {
6. @Cleanup InputStream in = **new** FileInputStream(args[0]);
7. @Cleanup OutputStream out = **new** FileOutputStream(args[1]);
8. **byte**[] b = **new** **byte**[10000];
9. **while** (**true**) {
10. **int** r = in.read(b);
11. **if** (r == -1) **break**;
12. out.write(b, 0, r);
13. }
14. }
15. }</span>

翻译后

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. <span style="font-weight:normal;">**import** java.io.\*;
3. **public** **class** CleanupExample {
4. **public** **static** **void** main(String[] args) **throws** IOException {
5. InputStream in = **new** FileInputStream(args[0]);
6. **try** {
7. OutputStream out = **new** FileOutputStream(args[1]);
8. **try** {
9. **byte**[] b = **new** **byte**[10000];
10. **while** (**true**) {
11. **int** r = in.read(b);
12. **if** (r == -1) **break**;
13. out.write(b, 0, r);
14. }
15. } **finally** {
16. **if** (out != **null**) {
17. out.close();
18. }
19. }
20. } **finally** {
21. **if** (in != **null**) {
22. in.close();
23. }
24. }
25. }
26. }</span>

#### 当然从1.7开始jdk已经提供了try with resources的方式自动回收资源

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **static** String readFirstLineFromFile(String path) **throws** IOException {
2. **try** (BufferedReader br = **new** BufferedReader(**new** FileReader(path))) {
3. **return** br.readLine();
4. }
5. }

#### **4.@Getter/@Setter**

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. <span style="font-weight:normal;">**import** lombok.AccessLevel;
2. **import** lombok.Getter;
3. **import** lombok.Setter;
5. **public** **class** GetterSetterExample {
6. /\*\*
7. \* Age of the person. Water is wet.
8. \*
9. \* @param age New value for this person's age. Sky is blue.
10. \* @return The current value of this person's age. Circles are round.
11. \*/
12. @Getter @Setter **private** **int** age = 10;
14. /\*\*
15. \* Name of the person.
16. \* -- SETTER --
17. \* Changes the name of this person.
18. \*
19. \* @param name The new value.
20. \*/
21. @Setter(AccessLevel.PROTECTED) **private** String name;
23. @Override **public** String toString() {
24. **return** String.format("%s (age: %d)", name, age);
25. }
26. }</span>

翻译后

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. <span style="font-weight:normal;"> **public** **class** GetterSetterExample {
2. /\*\*
3. \* Age of the person. Water is wet.
4. \*/
5. **private** **int** age = 10;
7. /\*\*
8. \* Name of the person.
9. \*/
10. **private** String name;
12. @Override **public** String toString() {
13. **return** String.format("%s (age: %d)", name, age);
14. }
16. /\*\*
17. \* Age of the person. Water is wet.
18. \*
19. \* @return The current value of this person's age. Circles are round.
20. \*/
21. **public** **int** getAge() {
22. **return** age;
23. }
25. /\*\*
26. \* Age of the person. Water is wet.
27. \*
28. \* @param age New value for this person's age. Sky is blue.
29. \*/
30. **public** **void** setAge(**int** age) {
31. **this**.age = age;
32. }
34. /\*\*
35. \* Changes the name of this person.
36. \*
37. \* @param name The new value.
38. \*/
39. **protected** **void** setName(String name) {
40. **this**.name = name;
41. }
42. }</span>

扩展配置：  
lombok.accessors.chain = [true | false] (default: false)如果设置为true，生成的setter将返回this(而不是void)，通过这个配置我们可以像jquery一样愉快的链式编程了。可以在类加增加一个@Accessors 注解 配置chain属性，优先于全局配置。  
lombok.accessors.fluent = [true | false] (default: false)如果设置为true，生成的getter和setter将不会使用bean标准的get、is或set进行前缀;相反，方法将使用与字段相同的名称(减去前缀)。可以在类加增加一个@Accessors注解，配置fluent属性，优先于全局配置  
lombok.accessors.prefix += a field prefix (default: empty list)给getter/setter方法增加前缀 例如配置 +=M 原有的 getFoo方法将变为getMFoo方法。 lombok.getter.noIsPrefix = [true | false] (default: false)如果设置为true，那么boolean型字段生成的getter将使用get前缀而不是默认的is前缀

### **5.@ToString**

### 生成一个toString方法，log debug神器

### 默认的toString格式为：ClassName(fieldName= fieleValue ,fieldName1=fieleValue)

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. <span style="font-weight:normal;">**import** lombok.ToString;
3. @ToString(exclude="id")
4. **public** **class** ToStringExample {
5. **private** **static** **final** **int** STATIC\_VAR = 10;
6. **private** String name;
7. **private** Shape shape = **new** Square(5, 10);
8. **private** String[] tags;
9. **private** **int** id;
11. **public** String getName() {
12. **return** **this**.getName();
13. }
15. @ToString(callSuper=**true**, includeFieldNames=**true**)
16. **public** **static** **class** Square **extends** Shape {
17. **private** **final** **int** width, height;
19. **public** Square(**int** width, **int** height) {
20. **this**.width = width;
21. **this**.height = height;
22. }
23. }
24. }</span>

翻译后

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** java.util.Arrays;
3. ublic **class** ToStringExample {
4. **private** **static** **final** **int** STATIC\_VAR = 10;
5. **private** String name;
6. **private** Shape shape = **new** Square(5, 10);
7. **private** String[] tags;
8. **private** **int** id;
10. **public** String getName() {
11. **return** **this**.getName();
12. }
14. **public** **static** **class** Square **extends** Shape {
15. **private** **final** **int** width, height;
17. **public** Square(**int** width, **int** height) {
18. **this**.width = width;
19. **this**.height = height;
20. }
22. @Override **public** String toString() {
23. **return** "Square(super=" + **super**.toString() + ", width=" + **this**.width + ", height=" + **this**.height + ")";
24. }
25. }
27. @Override **public** String toString() {
28. **return** "ToStringExample(" + **this**.getName() + ", " + **this**.shape + ", " + Arrays.deepToString(**this**.tags) + ")";
29. }

### 扩展配置：

### **lombok.toString.includeFieldNames**= [**true**|**false**] (default: true)****

通常，lombok以fieldName=fieldValue的形式为每个字段生成一个toString响应的片段。如果设置为false，lombok将省略字段的名称，可以在该注解上配置属性includeFieldNames来标示包含的字段，这样可以覆盖默认配置。

**lombok.toString.doNotUseGetters**= [**true**|**false**] (default: false)****  
如果设置为true，lombok将直接访问字段，而不是在生成tostring方法时使用getter(如果可用)。可以在该注解上配置属性doNotUseGetters来标示不使用getter的字段，这样可以覆盖默认配置。

### **6.@EqualsAndHashCode**

给类增加equals和hashCode方法

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.EqualsAndHashCode;
3. @EqualsAndHashCode(exclude={"id", "shape"})
4. **public** **class** EqualsAndHashCodeExample {
5. **private** **transient** **int** transientVar = 10;
6. **private** String name;
7. **private** **double** score;
8. **private** Shape shape = **new** Square(5, 10);
9. **private** String[] tags;
10. **private** **int** id;
12. **public** String getName() {
13. **return** **this**.name;
14. }
16. @EqualsAndHashCode(callSuper=**true**)
17. **public** **static** **class** Square **extends** Shape {
18. **private** **final** **int** width, height;
20. **public** Square(**int** width, **int** height) {
21. **this**.width = width;
22. **this**.height = height;
23. }
24. }
25. }

****翻译后****

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** java.util.Arrays;
3. ublic **class** EqualsAndHashCodeExample {
4. **private** **transient** **int** transientVar = 10;
5. **private** String name;
6. **private** **double** score;
7. **private** Shape shape = **new** Square(5, 10);
8. **private** String[] tags;
9. **private** **int** id;
11. **public** String getName() {
12. **return** **this**.name;
13. }
15. @Override **public** **boolean** equals(Object o) {
16. **if** (o == **this**) **return** **true**;
17. **if** (!(o **instanceof** EqualsAndHashCodeExample)) **return** **false**;
18. EqualsAndHashCodeExample other = (EqualsAndHashCodeExample) o;
19. **if** (!other.canEqual((Object)**this**)) **return** **false**;
20. **if** (**this**.getName() == **null** ? other.getName() != **null** : !**this**.getName().equals(other.getName())) **return** **false**;
21. **if** (Double.compare(**this**.score, other.score) != 0) **return** **false**;
22. **if** (!Arrays.deepEquals(**this**.tags, other.tags)) **return** **false**;
23. **return** **true**;
24. }
26. @Override **public** **int** hashCode() {
27. **final** **int** PRIME = 59;
28. **int** result = 1;
29. **final** **long** temp1 = Double.doubleToLongBits(**this**.score);
30. result = (result\*PRIME) + (**this**.name == **null** ? 43 : **this**.name.hashCode());
31. result = (result\*PRIME) + (**int**)(temp1 ^ (temp1 >>> 32));
32. result = (result\*PRIME) + Arrays.deepHashCode(**this**.tags);
33. **return** result;
34. }
36. **protected** **boolean** canEqual(Object other) {
37. **return** other **instanceof** EqualsAndHashCodeExample;
38. }
40. **public** **static** **class** Square **extends** Shape {
41. **private** **final** **int** width, height;
43. **public** Square(**int** width, **int** height) {
44. **this**.width = width;
45. **this**.height = height;
46. }
48. @Override **public** **boolean** equals(Object o) {
49. **if** (o == **this**) **return** **true**;
50. **if** (!(o **instanceof** Square)) **return** **false**;
51. Square other = (Square) o;
52. **if** (!other.canEqual((Object)**this**)) **return** **false**;
53. **if** (!**super**.equals(o)) **return** **false**;
54. **if** (**this**.width != other.width) **return** **false**;
55. **if** (**this**.height != other.height) **return** **false**;
56. **return** **true**;
57. }
59. @Override **public** **int** hashCode() {
60. **final** **int** PRIME = 59;
61. **int** result = 1;
62. result = (result\*PRIME) + **super**.hashCode();
63. result = (result\*PRIME) + **this**.width;
64. result = (result\*PRIME) + **this**.height;
65. **return** result;
66. }
68. **protected** **boolean** canEqual(Object other) {
69. **return** other **instanceof** Square;
70. }
71. }

****扩展配置：****

**lombok.config增加：**

lombok.equalsAndHashCode.doNotUseGetters = [true | false] (default: false)如果设置为true，lombok将直接访问字段，而不是在生成equals和hashcode方法时使用getter(如果可用)。  
可以在该注解上配置属性donotusegetter来标示不使用getter的字段，这样可以覆盖默认配置。  
lombok.equalsAndHashCode.callSuper = [call | skip | warn] (default: warn)如果设置为call，lombok将生成对hashCode的超类实现的调用。如果设置为skip，则不会生成这样的调用。默认行为warn类似于skip，并带有附加警告。  
  
  
  
****7.@NoArgsConstructor, @RequiredArgsConstructor and @AllArgsConstructor****

给类增加无参构造器 指定参数的构造器 包含所有参数的构造器

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.AccessLevel;
2. **import** lombok.RequiredArgsConstructor;
3. **import** lombok.AllArgsConstructor;
4. **import** lombok.NonNull;
6. @RequiredArgsConstructor(staticName = "of")
7. @AllArgsConstructor(access = AccessLevel.PROTECTED)
8. **public** **class** ConstructorExample<T> {
9. **private** **int** x, y;
10. @NonNull **private** T description;
12. @NoArgsConstructor
13. **public** **static** **class** NoArgsExample {
14. @NonNull **private** String field;
15. }
16. }

翻译后

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **public** **class** ConstructorExample<T> {
2. **private** **int** x, y;
3. @NonNull **private** T description;
5. **private** ConstructorExample(T description) {
6. **if** (description == **null**) **throw** **new** NullPointerException("description");
7. **this**.description = description;
8. }
10. **public** **static** <T> ConstructorExample<T> of(T description) {
11. **return** **new** ConstructorExample<T>(description);
12. }
14. @java.beans.ConstructorProperties({"x", "y", "description"})
15. **protected** ConstructorExample(**int** x, **int** y, T description) {
16. **if** (description == **null**) **throw** **new** NullPointerException("description");
17. **this**.x = x;
18. **this**.y = y;
19. **this**.description = description;
20. }
22. **public** **static** **class** NoArgsExample {
23. @NonNull **private** String field;
25. **public** NoArgsExample() {
26. }
27. }
28. }

扩展配置：

lombok.anyConstructor.suppressConstructorProperties = [true | false] (default: false)如果将其设置为true，那么lombok将跳过添加一个@java.bean.ConstructorProperties生成的构造器。这在android和GWT开发中很有用，因为这些注释通常不可用。

****8.@Data****

包含以下注解的集合

@ToString,@EqualsAndHashCode,所有字段的 @Getter 所有非final字段的@Setter ，@RequiredArgsConstructor

@EqualsAndHashCode

1. 此注解会生成equals(Object other) 和 hashCode()方法。

2. 它默认使用非静态，非瞬态的属性

3. 可通过参数exclude排除一些属性

4. 可通过参数of指定仅使用哪些属性

5. 它默认仅使用该类中定义的属性且不调用父类的方法

6. 可通过callSuper=true解决上一点问题。让其生成的方法中调用父类的方法。

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.AccessLevel;
2. **import** lombok.Setter;
3. **import** lombok.Data;
4. **import** lombok.ToString;
6. @Data **public** **class** DataExample {
7. **private** **final** String name;
8. @Setter(AccessLevel.PACKAGE) **private** **int** age;
9. **private** **double** score;
10. **private** String[] tags;
12. @ToString(includeFieldNames=**true**)
13. @Data(staticConstructor="of")
14. **public** **static** **class** Exercise<T> {
15. **private** **final** String name;
16. **private** **final** T value;
17. }
18. }

**翻译后**

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** java.util.Arrays;
3. **public** **class** DataExample {
4. **private** **final** String name;
5. **private** **int** age;
6. **private** **double** score;
7. **private** String[] tags;
9. **public** DataExample(String name) {
10. **this**.name = name;
11. }
13. **public** String getName() {
14. **return** **this**.name;
15. }
17. **void** setAge(**int** age) {
18. **this**.age = age;
19. }
21. **public** **int** getAge() {
22. **return** **this**.age;
23. }
25. **public** **void** setScore(**double** score) {
26. **this**.score = score;
27. }
29. **public** **double** getScore() {
30. **return** **this**.score;
31. }
33. **public** String[] getTags() {
34. **return** **this**.tags;
35. }
37. **public** **void** setTags(String[] tags) {
38. **this**.tags = tags;
39. }
41. @Override **public** String toString() {
42. **return** "DataExample(" + **this**.getName() + ", " + **this**.getAge() + ", " + **this**.getScore() + ", " + Arrays.deepToString(**this**.getTags()) + ")";
43. }
45. **protected** **boolean** canEqual(Object other) {
46. **return** other **instanceof** DataExample;
47. }
49. @Override **public** **boolean** equals(Object o) {
50. **if** (o == **this**) **return** **true**;
51. **if** (!(o **instanceof** DataExample)) **return** **false**;
52. DataExample other = (DataExample) o;
53. **if** (!other.canEqual((Object)**this**)) **return** **false**;
54. **if** (**this**.getName() == **null** ? other.getName() != **null** : !**this**.getName().equals(other.getName())) **return** **false**;
55. **if** (**this**.getAge() != other.getAge()) **return** **false**;
56. **if** (Double.compare(**this**.getScore(), other.getScore()) != 0) **return** **false**;
57. **if** (!Arrays.deepEquals(**this**.getTags(), other.getTags())) **return** **false**;
58. **return** **true**;
59. }
61. @Override **public** **int** hashCode() {
62. **final** **int** PRIME = 59;
63. **int** result = 1;
64. **final** **long** temp1 = Double.doubleToLongBits(**this**.getScore());
65. result = (result\*PRIME) + (**this**.getName() == **null** ? 43 : **this**.getName().hashCode());
66. result = (result\*PRIME) + **this**.getAge();
67. result = (result\*PRIME) + (**int**)(temp1 ^ (temp1 >>> 32));
68. result = (result\*PRIME) + Arrays.deepHashCode(**this**.getTags());
69. **return** result;
70. }
72. **public** **static** **class** Exercise<T> {
73. **private** **final** String name;
74. **private** **final** T value;
76. **private** Exercise(String name, T value) {
77. **this**.name = name;
78. **this**.value = value;
79. }
81. **public** **static** <T> Exercise<T> of(String name, T value) {
82. **return** **new** Exercise<T>(name, value);
83. }
85. **public** String getName() {
86. **return** **this**.name;
87. }
89. **public** T getValue() {
90. **return** **this**.value;
91. }
93. @Override **public** String toString() {
94. **return** "Exercise(name=" + **this**.getName() + ", value=" + **this**.getValue() + ")";
95. }
97. **protected** **boolean** canEqual(Object other) {
98. **return** other **instanceof** Exercise;
99. }
101. @Override **public** **boolean** equals(Object o) {
102. **if** (o == **this**) **return** **true**;
103. **if** (!(o **instanceof** Exercise)) **return** **false**;
104. Exercise<?> other = (Exercise<?>) o;
105. **if** (!other.canEqual((Object)**this**)) **return** **false**;
106. **if** (**this**.getName() == **null** ? other.getValue() != **null** : !**this**.getName().equals(other.getName())) **return** **false**;
107. **if** (**this**.getValue() == **null** ? other.getValue() != **null** : !**this**.getValue().equals(other.getValue())) **return** **false**;
108. **return** **true**;
109. }
111. @Override **public** **int** hashCode() {
112. **final** **int** PRIME = 59;
113. **int** result = 1;
114. result = (result\*PRIME) + (**this**.getName() == **null** ? 43 : **this**.getName().hashCode());
115. result = (result\*PRIME) + (**this**.getValue() == **null** ? 43 : **this**.getValue().hashCode());
116. **return** result;
117. }
118. }
119. }

9.@Value

**@value是@data的不可变对象 （不可变对象的用处和创建：**[https://my.oschina.net/jasonultimate/blog/166810](https://my.oschina.net/jasonultimate/blog/166810" \t "http://blog.csdn.net/v2sking/article/details/_blank)**）**

**所有字段都是私有的，默认情况下是final的，并且不会生成setter。默认情况下，类本身也是final的，因为不可变性不能强制转化为子类。与@data一样，有用toString()、equals()和hashCode()方法也是生成的，每个字段都有一个getter方法，并且一个覆盖每个参数的构造器也会生成。**

**10.@Builder**

**建筑者模式**

**是现在比较推崇的一种构建值对象的方式。**

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.Builder;
2. **import** lombok.Singular;
3. **import** java.util.Set;
5. @Builder
6. **public** **class** BuilderExample {
7. **private** String name;
8. **private** **int** age;
9. @Singular **private** Set<String> occupations;
10. }

****翻译后****

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** java.util.Set;
3. **public** **class** BuilderExample {
4. **private** String name;
5. **private** **int** age;
6. **private** Set<String> occupations;
8. BuilderExample(String name, **int** age, Set<String> occupations) {
9. **this**.name = name;
10. **this**.age = age;
11. **this**.occupations = occupations;
12. }
14. **public** **static** BuilderExampleBuilder builder() {
15. **return** **new** BuilderExampleBuilder();
16. }
18. **public** **static** **class** BuilderExampleBuilder {
19. **private** String name;
20. **private** **int** age;
21. **private** java.util.ArrayList<String> occupations;
23. BuilderExampleBuilder() {
24. }
26. **public** BuilderExampleBuilder name(String name) {
27. **this**.name = name;
28. **return** **this**;
29. }
31. **public** BuilderExampleBuilder age(**int** age) {
32. **this**.age = age;
33. **return** **this**;
34. }
36. **public** BuilderExampleBuilder occupation(String occupation) {
37. **if** (**this**.occupations == **null**) {
38. **this**.occupations = **new** java.util.ArrayList<String>();
39. }
41. **this**.occupations.add(occupation);
42. **return** **this**;
43. }
45. **public** BuilderExampleBuilder occupations(Collection<? **extends** String> occupations) {
46. **if** (**this**.occupations == **null**) {
47. **this**.occupations = **new** java.util.ArrayList<String>();
48. }
50. **this**.occupations.addAll(occupations);
51. **return** **this**;
52. }
54. **public** BuilderExampleBuilder clearOccupations() {
55. **if** (**this**.occupations != **null**) {
56. **this**.occupations.clear();
57. }
59. **return** **this**;
60. }
62. **public** BuilderExample build() {
63. // complicated switch statement to produce a compact properly sized immutable set omitted.
64. // go to https://projectlombok.org/features/Singular-snippet.html to see it.
65. Set<String> occupations = ...;
66. **return** **new** BuilderExample(name, age, occupations);
67. }
69. @java.lang.Override
70. **public** String toString() {
71. **return** "BuilderExample.BuilderExampleBuilder(name = " + **this**.name + ", age = " + **this**.age + ", occupations = " + **this**.occupations + ")";
72. }
73. }
74. }

**11.@SneakyThrows**

把checked异常转化为unchecked异常，好处是不用再往上层方法抛出了，美其名曰暗埋异常

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.SneakyThrows;
3. **public** **class** SneakyThrowsExample **implements** Runnable {
4. @SneakyThrows(UnsupportedEncodingException.**class**)
5. **public** String utf8ToString(**byte**[] bytes) {
6. **return** **new** String(bytes, "UTF-8");
7. }
9. @SneakyThrows
10. **public** **void** run() {
11. **throw** **new** Throwable();
12. }
13. }

****翻译后：****

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.Lombok;
3. **public** **class** SneakyThrowsExample **implements** Runnable {
4. **public** String utf8ToString(**byte**[] bytes) {
5. **try** {
6. **return** **new** String(bytes, "UTF-8");
7. } **catch** (UnsupportedEncodingException e) {
8. **throw** Lombok.sneakyThrow(e);
9. }
10. }
12. **public** **void** run() {
13. **try** {
14. **throw** **new** Throwable();
15. } **catch** (Throwable t) {
16. **throw** Lombok.sneakyThrow(t);
17. }
18. }
19. }

12.@Synchronized

类似于Synchronized 关键字 但是可以隐藏同步锁

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.Synchronized;
3. ublic **class** SynchronizedExample {
4. **private** **final** Object readLock = **new** Object();
6. @Synchronized
7. **public** **static** **void** hello() {
8. System.out.println("world");
9. }
11. @Synchronized
12. **public** **int** answerToLife() {
13. **return** 42;
14. }
16. @Synchronized("readLock")
17. **public** **void** foo() {
18. System.out.println("bar");
19. }

****翻译后****

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **public** **class** SynchronizedExample {
2. **private** **static** **final** Object $LOCK = **new** Object[0];
3. **private** **final** Object $lock = **new** Object[0];
4. **private** **final** Object readLock = **new** Object();
6. **public** **static** **void** hello() {
7. **synchronized**($LOCK) {
8. System.out.println("world");
9. }
10. }
12. **public** **int** answerToLife() {
13. **synchronized**($lock) {
14. **return** 42;
15. }
16. }
18. **public** **void** foo() {
19. **synchronized**(readLock) {
20. System.out.println("bar");
21. }
22. }
23. }

****xianzjdk推荐使用Lock了，这个仅供参考****

**13.@Getter(lazy=true)**

如果getter方法计算值需要大量CPU，或者值占用大量内存，第一次调用这个getter，它将一次计算一个值，然后从那时开始缓存它

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.Getter;
3. **public** **class** GetterLazyExample {
4. @Getter(lazy=**true**) **private** **final** **double**[] cached = expensive();
6. **private** **double**[] expensive() {
7. **double**[] result = **new** **double**[1000000];
8. **for** (**int** i = 0; i < result.length; i++) {
9. result[i] = Math.asin(i);
10. }
11. **return** result;
12. }
13. }

****翻译后****

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **public** **class** GetterLazyExample {
2. **private** **final** java.util.concurrent.AtomicReference<java.lang.Object> cached = **new** java.util.concurrent.AtomicReference<java.lang.Object>();
4. **public** **double**[] getCached() {
5. java.lang.Object value = **this**.cached.get();
6. **if** (value == **null**) {
7. **synchronized**(**this**.cached) {
8. value = **this**.cached.get();
9. **if** (value == **null**) {
10. **final** **double**[] actualValue = expensive();
11. value = actualValue == **null** ? **this**.cached : actualValue;
12. **this**.cached.set(value);
13. }
14. }
15. }
16. **return** (**double**[])(value == **this**.cached ? **null** : value);
17. }
19. **private** **double**[] expensive() {
20. **double**[] result = **new** **double**[1000000];
21. **for** (**int** i = 0; i < result.length; i++) {
22. result[i] = Math.asin(i);
23. }
24. **return** result;
25. }

**14.@Log**

**可以生成各种log对象，方便多了**

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **import** lombok.extern.java.Log;
2. **import** lombok.extern.slf4j.Slf4j;
4. @Log
5. **public** **class** LogExample {
7. **public** **static** **void** main(String... args) {
8. log.error("Something's wrong here");
9. }
10. }
12. @Slf4j
13. **public** **class** LogExampleOther {
15. **public** **static** **void** main(String... args) {
16. log.error("Something else is wrong here");
17. }
18. }
20. @CommonsLog(topic="CounterLog")
21. **public** **class** LogExampleCategory {
23. **public** **static** **void** main(String... args) {
24. log.error("Calling the 'CounterLog' with a message");
25. }
26. }

****翻译为****

**[java]** [view plain](http://blog.csdn.net/v2sking/article/details/73431364" \o "view plain) [copy](http://blog.csdn.net/v2sking/article/details/73431364" \o "copy)

1. **public** **class** LogExample {
2. **private** **static** **final** java.util.logging.Logger log = java.util.logging.Logger.getLogger(LogExample.**class**.getName());
4. **public** **static** **void** main(String... args) {
5. log.error("Something's wrong here");
6. }
7. }
9. **public** **class** LogExampleOther {
10. **private** **static** **final** org.slf4j.Logger log = org.slf4j.LoggerFactory.getLogger(LogExampleOther.**class**);
12. **public** **static** **void** main(String... args) {
13. log.error("Something else is wrong here");
14. }
15. }
17. **public** **class** LogExampleCategory {
18. **private** **static** **final** org.apache.commons.logging.Log log = org.apache.commons.logging.LogFactory.getLog("CounterLog");
20. **public** **static** **void** main(String... args) {
21. log.error("Calling the 'CounterLog' with a message");
22. }
23. }

**所有支持的log类型：**

@CommonsLog Createsprivate static final [org.apache.commons.logging.Log](https://commons.apache.org/logging/apidocs/org/apache/commons/logging/Log.html" \t "http://blog.csdn.net/v2sking/article/details/_blank) log = [org.apache.commons.logging.LogFactory.getLog](https://commons.apache.org/logging/apidocs/org/apache/commons/logging/LogFactory.html" \l "getLog(java.lang.Class)" \t "http://blog.csdn.net/v2sking/article/details/_blank)(LogExample.class); @JBossLog Createsprivate static final [org.jboss.logging.Logger](http://docs.jboss.org/jbosslogging/latest/org/jboss/logging/Logger.html" \t "http://blog.csdn.net/v2sking/article/details/_blank) log = [org.jboss.logging.Logger.getLogger](http://docs.jboss.org/jbosslogging/latest/org/jboss/logging/Logger.html" \l "getLogger(java.lang.Class)" \t "http://blog.csdn.net/v2sking/article/details/_blank)(LogExample.class); @Log Createsprivate static final [java.util.logging.Logger](http://download.oracle.com/javase/6/docs/api/java/util/logging/Logger.html" \t "http://blog.csdn.net/v2sking/article/details/_blank) log = [java.util.logging.Logger.getLogger](http://download.oracle.com/javase/6/docs/api/java/util/logging/Logger.html" \l "getLogger(java.lang.String)" \t "http://blog.csdn.net/v2sking/article/details/_blank)(LogExample.class.getName()); @Log4j Createsprivate static final [org.apache.log4j.Logger](https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/Logger.html" \t "http://blog.csdn.net/v2sking/article/details/_blank) log = [org.apache.log4j.Logger.getLogger](https://logging.apache.org/log4j/1.2/apidocs/org/apache/log4j/Logger.html" \l "getLogger(java.lang.Class)" \t "http://blog.csdn.net/v2sking/article/details/_blank)(LogExample.class); @Log4j2 Createsprivate static final [org.apache.logging.log4j.Logger](https://logging.apache.org/log4j/2.0/log4j-api/apidocs/org/apache/logging/log4j/Logger.html" \t "http://blog.csdn.net/v2sking/article/details/_blank) log = [org.apache.logging.log4j.LogManager.getLogger](https://logging.apache.org/log4j/2.0/log4j-api/apidocs/org/apache/logging/log4j/LogManager.html" \l "getLogger(java.lang.Class)" \t "http://blog.csdn.net/v2sking/article/details/_blank)(LogExample.class); @Slf4j Creates private static final [org.slf4j.Logger](http://www.slf4j.org/api/org/slf4j/Logger.html" \t "http://blog.csdn.net/v2sking/article/details/_blank) log = [org.slf4j.LoggerFactory.getLogger](http://www.slf4j.org/apidocs/org/slf4j/LoggerFactory.html" \l "getLogger(java.lang.Class)" \t "http://blog.csdn.net/v2sking/article/details/_blank)(LogExample.class); @XSlf4j Createsprivate static final [org.slf4j.ext.XLogger](http://www.slf4j.org/api/org/slf4j/ext/XLogger.html" \t "http://blog.csdn.net/v2sking/article/details/_blank) log = [org.slf4j.ext.XLoggerFactory.getXLogger](http://www.slf4j.org/apidocs/org/slf4j/ext/XLoggerFactory.html" \l "getXLogger(java.lang.Class)" \t "http://blog.csdn.net/v2sking/article/details/_blank)(LogExample.class);

扩展配置：

lombok.log.fieldName = an identifier (default: log).生成log字段的名称 默认为log lombok.log.fieldIsStatic = [true | false] (default: true)生成log是否是static的 默认为static

官方文档说明：[https://projectlombok.org/features/all](https://projectlombok.org/features/all" \t "http://blog.csdn.net/v2sking/article/details/_blank)

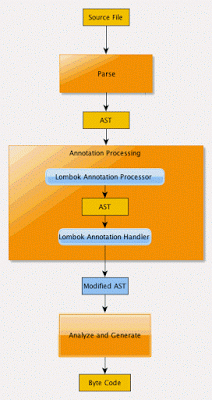
# Lombok原理

lombok通过简单的注解标志就能够实现复杂的代码生成，他是怎么做到的？

lombok注解不是我们常见的runtime注解，而是source注解或者class注解，

在没有jsr之前我们可以通过反射在运行是获取注解值，但是这样效率很低，而且没办法做到编译检查，对开发人员一些不合的编码错误给出警告，

JSR 269: Pluggable Annotation Processing API ([https://www.jcp.org/en/jsr/detail?id=269](https://www.jcp.org/en/jsr/detail?id=269" \t "http://blog.csdn.net/v2sking/article/details/_blank)) 出现后，我们可以在javac的编译器利用注解来完成对class文件的修改。



lombok本质上就是这样的一个实现了"JSR 269 API"的程序。在使用javac的过程中，它产生作用的具体流程如下：  
1)javac对源代码进行分析，生成一棵抽象语法树(AST)  
2)运行过程中调用实现了"JSR 269 API"的lombok程序  
3)此时lombok就对第一步骤得到的AST进行处理，找到@Data注解所在类对应的语法树(AST)，然后修改该语法树(AST)，增加getter和setter方法定义的相应树节点  
4)javac使用修改后的抽象语法树(AST)生成字节码文件，即给class增加新的节点（代码块）