## 一、Java注解编程

在开发过程中我们会看到，继承接口会出现@Override注解，有时候还会提示写注解@SuppressWarnings  
还有在使用**[spring](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "/home/caojx/文档\\x/_blank)**,Mybatis，Struts2等框架时，常常会使用注解，很多人都知道这些怎么使用，但是不知道他是怎么来的。其实，这是**[Java](http://lib.csdn.net/base/javase" \o "Java SE知识库" \t "/home/caojx/文档\\x/_blank)**的特性

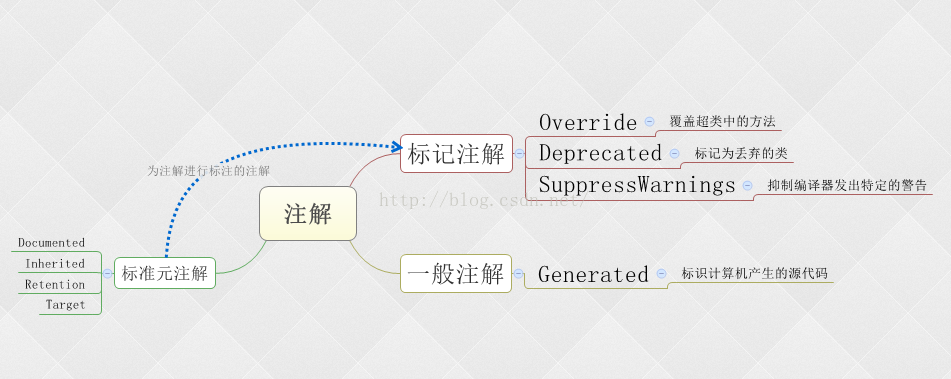
通过学习注解编程我们可以

1.能够读懂别人写的代码，特别是框架相关的代码  
2.编程更加简洁，代码更加清晰

Java注解是一种代码级别的说明。它是JDK1.5及以后版本引入的一个特性，与类、接口、枚举是在同一个层次。它可以声明在包、类、字段、方法、局部变量、方法参数等的前面，用来对这些元素进行说明，注释。

## 二、注解的分类

这里引用网友的一张图：http://www.cnblogs.com/xing901022/p/3966799.html



上图分类可以分为三类：注解大体上分为三种：标记注解，一般注解，元注解，也可以按照如下机制分类

#### 2.1按照运机制分为

源码注解  
编译时的注解  
运行时的注解

源码注解：注解只在源码存在，编译成.class文件就不存在了  
编译时注解：注解在源码和.class中都存在 （如@Override,@Deprecated,@Suppvisewarnings）  
运行时注解：在运行阶段还会起作用，甚至会影响运行逻辑的注解（@Autowired）

#### 2.2按照注解的来源

来自JDK的注解  
来自第三方的注解  
自定义的注解

#### 2.3元注解

给注解进行的注解

## 三、常见的注解实例

@PostConstruct 依赖资源加载后执行该注解所注解的方法

@Override  //实现类实现接口的时候常常会见到，说明覆盖了父类或接口中的方法  
@Deprecated //声明某个方法过时了  
@Suppvisewarnings //忽略警告,抑制编译器发出特定的警告

@Override标识覆盖父类或父接口中的方法

实现接口或继承父类的时候，方法上就会有@Override,表示覆盖父类，或父接口中的方法

@Deprecated 标注某个方法过时了，不建议使用

@Suppvisewarnings 忽略警告,抑制编译器发出特定的警告

在我们调用过时的方法或其他情况下我们会看到编译器给我们发出告警

我们可以使用@Suppvisewarnings("抑制那种告警的注解名")，如下，告警消失了

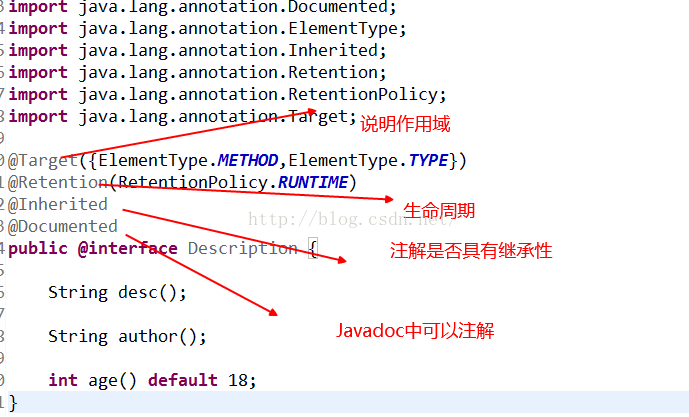
还有一些常见的第三方框架的注解

Spring    @Autowired  @Service  @Repository    
Mybatis   @InserProvider @UpdateProvier  @Options

## 四、自定义注解

我们主要了解如下内容：

自定义注解的语法要求  
注解的注解（元注解）  
使用自定义注解  
解析注解



#### 4.1自定义注解的语法要求

自定义注解有如下要求：

1.成员类型是受限制的，合法的类型包括原始类型(基本数据类型)以及String,Class,Annotaion,Enumeration

2.注解可以设置默认值，如上图 defalut 18

3.如果只有个注解成员，成员应该为 String value();  //标准是这样建议的

4.注解成员必须是无参无异常方式声明

5.注解类可以没有成员，没有成员的注解称为标识注解

#### 4.2注解的注解（元注解）

上图中的注解类中的注解都是元注解

@Target : 表示该注解的作用范围，可能的值在枚举类 ElemenetType 中，包括：   
          ElemenetType.CONSTRUCTOR        构造器声明   
          ElemenetType.FIELD          　　域声明（包括 enum 实例）   
          ElemenetType.LOCAL\_VARIABLE     局部变量声明   
          ElemenetType.METHOD           　方法声明   
          ElemenetType.PACKAGE          　包声明   
          ElemenetType.PARAMETER          参数声明   
          ElemenetType.TYPE          　　 类，接口（包括注解类型）或enum声明  
@Retention : 表示在什么级别保存该注解信息。可选的参数值在枚举类型 RetentionPolicy 中，包括：   
          RetentionPolicy.SOURCE 　　　    注解将被编译器丢弃   
          RetentionPolicy.CLASS 　　　　　 注解在class文件中可用，但会被VM丢弃   
          RetentionPolicy.RUNTIME 　　　　 VM将在运行期也保留注释，因此可以通过反射机制读取注解的信息。  
@Documented : 将此注解包含在 javadoc 中 ，它代表着此注解会被javadoc工具提取成文档。在doc文档中的内容会因为此注解的信息内容不同而不同。  
@Inherited ：注解类中使用该注解标识，如果类中使用了该注解，子类会继承类中（只有在继承的时候生效，而且只会继承类中的注解，方法或属性中的注解不会生效）的注解

## 五、注解的解析

概念：通过反射获取类，函数或成员上的运行时的注解信息，从而实现动态控制程序运行的逻辑

自定义注解类：

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/53001224) [copy](http://blog.csdn.net/u014292162/article/details/53001224) [IMG_258](https://code.csdn.net/snippets/1961757)[IMG_259](https://code.csdn.net/snippets/1961757/fork)

1. @Target({ElementType.METHOD,ElementType.TYPE})
2. @Retention(RetentionPolicy.RUNTIME)
3. @Inherited
4. @Documented
5. **public** **@interface** Description {
7. String value();
8. }

Person.java

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/53001224) [copy](http://blog.csdn.net/u014292162/article/details/53001224) [IMG_260](https://code.csdn.net/snippets/1961757)[IMG_261](https://code.csdn.net/snippets/1961757/fork)

1. **package** com.ann.test;
3. @Description("I am a class annotation")
4. **public** **class** Person {
6. @Description("I am a method annotation")
7. **public** String name() {
8. **return** **null**;
9. }
11. @Description("I am a method annotation")
12. **public** **int** age(){
13. **return** 0;
14. }
16. @Deprecated  //注解说明该方法过时了
17. **public** **void** sing(){
19. }
20. }

Child.java，继承Person，验证@Inherited

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/53001224) [copy](http://blog.csdn.net/u014292162/article/details/53001224) [IMG_262](https://code.csdn.net/snippets/1961757)[IMG_263](https://code.csdn.net/snippets/1961757/fork)

1. **package** com.ann.test;
3. **public** **class** Child **extends** Person {
5. @Override
6. **public** String name() {
7. **return** **null**;
8. }
10. @Override
11. **public** **int** age() {
12. **return** 0;
13. }
15. @Override
16. **public** **void** sing() {
18. }
20. }

ParseAnn.java解析注解，主要通过反射解析

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/53001224) [copy](http://blog.csdn.net/u014292162/article/details/53001224) [IMG_264](https://code.csdn.net/snippets/1961757)[IMG_265](https://code.csdn.net/snippets/1961757/fork)

1. **package** com.ann.test;
3. **import** java.lang.annotation.Annotation;
4. **import** java.lang.reflect.Method;
6. /\*
7. \* 通过反射解析注解
8. \* \*/
9. **public** **class** ParseAnn {
11. **public** **static** **void** main(String[] args) {
12. //1.使用类加载器加载类
13. **try**{
14. Class c = Class.forName("com.ann.test.Child");
15. //2.找到类上边的注解
16. **boolean** isExist = c.isAnnotationPresent(Description.**class**);
17. **if**(isExist){
18. //3.拿到注解实例
19. Description description = (Description) c.getAnnotation(Description.**class**);
20. System.out.println(description.value()+"---1");
21. }
23. //4.找到方法上的注解
24. Method[] methods =c.getMethods();
25. **for**(Method method:methods){
26. **boolean** isMExist = method.isAnnotationPresent(Description.**class**);
27. **if**(isMExist){
28. //5.拿到方法上的注解实例
29. Description dmethod = method.getAnnotation(Description.**class**);
30. System.out.println(dmethod.value()+"---2");
31. }
32. }
34. //获取所有的方法上的注解
35. **for**(Method m:methods){
36. Annotation[] annotations = m.getAnnotations();
37. **for**(Annotation annotation:annotations){
38. **if**(annotation **instanceof** Description){
39. Description d = (Description)annotation;
40. System.out.println(d.value()+"---3");
41. }
42. }
43. }
44. }**catch**(ClassNotFoundException e){
45. e.printStackTrace();
46. }
47. }
49. }

结果：

I am a class annotation---1

从结果可以看出，只有类上的注解被继承类，通过反实现一个仿照**[hibernate](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "/home/caojx/文档\\x/_blank)**的解决方案，核心代码通过注解实现

## 六、仿照Hibernate自定义注解实现

需求：1.有一张用户表，字段包括用户ID,用户名，昵称，年龄，性别，所在城市，邮箱，手机号。  
2.为方便对每个字段的组合条件进行检索，并打印出SQL  
3.使用方式足够简单射可以很方便的解析出注解

自定义注解类：用于注解表Table

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/53001224) [copy](http://blog.csdn.net/u014292162/article/details/53001224) [IMG_266](https://code.csdn.net/snippets/1961757)[IMG_267](https://code.csdn.net/snippets/1961757/fork)

1. **package** com.ann.test2;
3. **import** java.lang.annotation.ElementType;
4. **import** java.lang.annotation.Retention;
5. **import** java.lang.annotation.RetentionPolicy;
6. **import** java.lang.annotation.Target;
8. @Target({ElementType.TYPE})
9. @Retention(RetentionPolicy.RUNTIME)
10. **public** **@interface** Table {
12. String value();
13. }

自定义注解类：用于注解Column

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/53001224) [copy](http://blog.csdn.net/u014292162/article/details/53001224) [IMG_268](https://code.csdn.net/snippets/1961757)[IMG_269](https://code.csdn.net/snippets/1961757/fork)

1. **import** java.lang.annotation.ElementType;
2. **import** java.lang.annotation.Retention;
3. **import** java.lang.annotation.RetentionPolicy;
4. **import** java.lang.annotation.Target;
6. @Target({ElementType.FIELD})
7. @Retention(RetentionPolicy.RUNTIME)
8. **public** **@interface** Column {
10. String value();
11. }

注解的使用：

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/53001224) [copy](http://blog.csdn.net/u014292162/article/details/53001224) [IMG_270](https://code.csdn.net/snippets/1961757)[IMG_271](https://code.csdn.net/snippets/1961757/fork)

1. **package** com.ann.test2;
3. @Table("user")
4. **public** **class** Filter {
6. @Column("id")
7. **private** **int** id;
9. @Column("user\_name")
10. **private** String userName;
12. @Column("nick\_name")
13. **private** String nickName;
15. @Column("age")
16. **private** **int** age;
18. @Column("city")
19. **private** String city;
21. @Column("email")
22. **private** String email;
24. @Column("mobile")
25. **private** String mobile;
27. **public** **int** getId() {
28. **return** id;
29. }
31. **public** **void** setId(**int** id) {
32. **this**.id = id;
33. }
35. **public** String getUserName() {
36. **return** userName;
37. }
39. **public** **void** setUserName(String userName) {
40. **this**.userName = userName;
41. }
43. **public** String getNickName() {
44. **return** nickName;
45. }
47. **public** **void** setNickName(String nickName) {
48. **this**.nickName = nickName;
49. }
51. **public** **int** getAge() {
52. **return** age;
53. }
55. **public** **void** setAge(**int** age) {
56. **this**.age = age;
57. }
59. **public** String getCity() {
60. **return** city;
61. }
63. **public** **void** setCity(String city) {
64. **this**.city = city;
65. }
67. **public** String getEmail() {
68. **return** email;
69. }
71. **public** **void** setEmail(String email) {
72. **this**.email = email;
73. }
75. **public** String getMobile() {
76. **return** mobile;
77. }
79. **public** **void** setMobile(String mobile) {
80. **this**.mobile = mobile;
81. }
83. }

注解解析成sql

**[java]** [view plain](http://blog.csdn.net/u014292162/article/details/53001224) [copy](http://blog.csdn.net/u014292162/article/details/53001224) [IMG_272](https://code.csdn.net/snippets/1961757)[IMG_273](https://code.csdn.net/snippets/1961757/fork)

1. **package** com.ann.test2;
3. **import** java.io.File;
4. **import** java.lang.reflect.Field;
5. **import** java.lang.reflect.InvocationTargetException;
6. **import** java.lang.reflect.Method;
8. **public** **class** Test {
10. **public** **static** **void** main(String[] args) {
11. Filter filter = **new** Filter();
12. filter.setId(10);//查询id为10的用户
14. Filter filter2 = **new** Filter();
15. filter2.setUserName("lucy"); //查询用户名为lucy的用户
17. Filter filter3 = **new** Filter();
18. filter3.setEmail("liu@sina.com,zh@163.com,88888@qq.com");
20. String sql1= query(filter);
21. String sql2= query(filter2);
22. String sql3= query(filter3);
24. System.out.println(sql1);
25. System.out.println(sql2);
26. System.out.println(sql3);
27. }
29. **private** **static** String query(Filter filter){
30. StringBuilder sb = **new** StringBuilder();
31. //1.获取class
32. Class c = filter.getClass();
33. //2.获取到Table的名字
34. **boolean** exists = c.isAnnotationPresent(Table.**class**);
35. **if**(!exists){
36. **return** **null**;
37. }
38. Table table = (Table) c.getAnnotation(Table.**class**);
39. String tableName = table.value();
40. sb.append("select \* form ").append(tableName).append(" where 1=1 ");
41. //3.遍历所有的字段
42. Field[] fArray = c.getDeclaredFields();
43. **for**(Field f:fArray){
44. //4.处理每个字段对应的sql
45. //4.1拿到字段的名字
46. **boolean** fExists = f.isAnnotationPresent(Column.**class**);
47. **if**(!fExists){
48. **continue**;
49. }
50. Column column = f.getAnnotation(Column.**class**);
51. //4.2拿到字段的值
52. String columnName = column.value();
53. //获取属性的值
54. String filedName = f.getName();
55. String getMethodName = "get"+filedName.substring(0, 1).toUpperCase()+filedName.substring(1);
56. **try** {
57. Method method = c.getMethod(getMethodName);
58. Object fileValue =method.invoke(filter);
59. //4.3拼接sql
60. **if**(fileValue==**null** || (fileValue **instanceof** Integer &&(Integer)fileValue==0)){
61. **continue**;
62. }
63. sb.append(" and ").append(filedName);
64. **if**(fileValue **instanceof** String){// in()
65. **if**(((String) fileValue).contains(",")){
66. String[] values = ((String)fileValue).split(",");
67. sb.append(" in (");
68. **for**(String v:values){
69. sb.append("'").append(v).append("'").append(",");
70. }
71. sb.deleteCharAt(sb.length()-1);
72. sb.append(")");
74. }**else**{
75. sb.append("=").append("'").append(fileValue).append("'");
76. }
77. }**else** **if**(fileValue **instanceof** Integer){
78. sb.append("=").append(fileValue);
79. }
80. } **catch** (Exception e) {
81. e.printStackTrace();
82. }
83. }
84. **return** sb.toString();
85. }
86. }

结果：

select \* form user where 1=1  and id=10  
select \* form user where 1=1  and userName='lucy'  
select \* form user where 1=1  and email in ('liu@sina.com','zh@163.com','88888@qq.com')

还可以参见博客：http://www.cnblogs.com/pepcod/archive/2013/02/20/2918719.html