注解的解析

1. 首先要关注注解的分类
   1. 按运行机制分：
      1. 源码注解（编译器丢弃的注释）

RetentionPolicy.SOURCE

* + 1. 编译时注解（编译器把注释记录在class文件中，运行时VM不需要保留注释）

RetentionPolicy.CLASS

* + 1. 运行时注解（编译器把注释记录在class文件中，运行时VM保留注释，可以反复使用）

RetentionPolicy.RUNTIME

在定义注解时使用元注解规定生命周期，以上三者用@Retention

@Retention(RUNTIME)或者@Retention(RetentionPolicy.RUNTIME)

* 1. 按来源分：
     1. JDK原生注解
     2. 第三方注解
     3. 自定义注解

1. 自定义注解语法

@Target({ElementType.METHOD,ElementType.TYPE})

@Retention(RUNTIME)

@Inherited

@Documented

public @interface Description{

String desc();

String author();

int age() default 18;

}

以上四个是元注解，我们在使用@interface定义注解时可以使用元注解进行注解

1）@Target({ElementType.METHOD,ElementType.TYPE})

@Target表示注解的使用范围

@Target(ElementType.TYPE)   //接口、类、枚举、注解

@Target(ElementType.FIELD) //字段、枚举的常量

@Target(ElementType.METHOD) //方法

@Target(ElementType.PARAMETER) //方法参数

@Target(ElementType.CONSTRUCTOR)  //构造函数

@Target(ElementType.LOCAL\_VARIABLE)//局部变量

@Target(ElementType.ANNOTATION\_TYPE)//注解

@Target(ElementType.PACKAGE) ///包

2）@Retention(RUNTIME)

@Retention表示注解的生命周期，1.已有，不赘述

3）@Inherited

表示被注解的类的子类也会自动使用这个注解

4）@Documented

表示这个注解在生成javadoc时也会生成

1. 用反射来解析注解上填写的值
2. 注解没什么特殊的，和大多数XML一样只是作为配置文件存在！，也一样需要外界的程序来进行解析，里面的值也一样是固定的
3. 注解一般使用反射来解析，而反射是运行时技术，因此只有@Retention为RUNTIME的才能解析

而注解在类中是field类型的（@Target声明为ElementType.FIELD，也注解了field的），使用field.isAnnotationPresent(Column.class)来判断某个field是否Column注解，

如果是的话，使用field.getAnnotation(Column.class)来返回注解Column对象，通过这个注解的原型我们可以返回注解的值 column.value()，返回的是Object类型，可强转成对应类型

**总结**：具体的Method也好、Field也好还是Class也好等等其他的@Target中声明可注解的位置，要取得这个注解时，只要在反射的时候调用用对应对象（Method、Field。。。）的getAnnotation或getAnnotations方法（返回Annotation数组）。

3）其余的就是业务逻辑了。具体可以看另一份word中关于注解实现字段映射的。