# 反射机制（通过类对象获取类信息）

反射的意思就是通过类对象来获取类信息（方法，属性）的意思，

一定注意，是通过类对象去获取，类对象是关键。

三个大字：类对象

## 类对象的概念

咱们普通的new出来的对象，比如盖伦和亚索，都是基于Hero类模板new出来的。

类对象指的就是那个模板Hero。

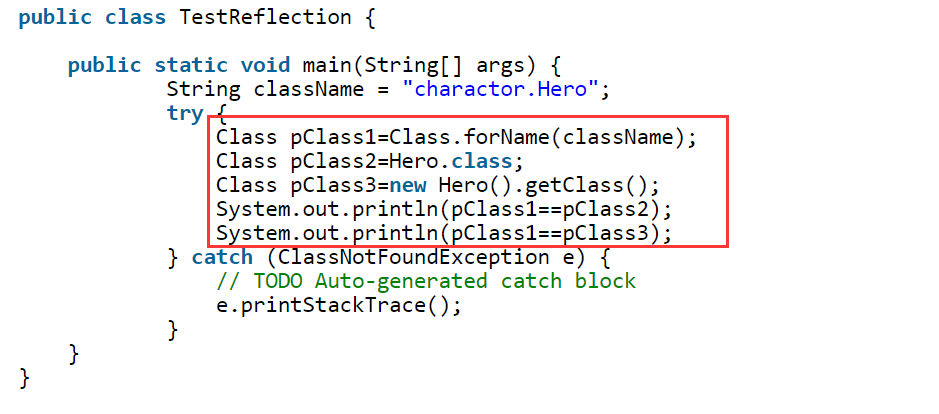
1.类对象有三种取得的方式：

1.Class.forName(“xxx”) //里面是字符串，表示那个类对象的路径。如charactor.Hero，这种方式多用于配置文件对内容的控制。

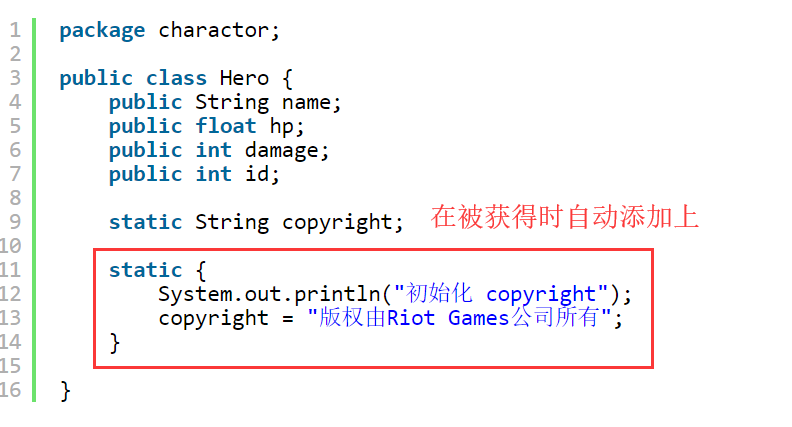
2. Hero.class 这种方式。（常见）

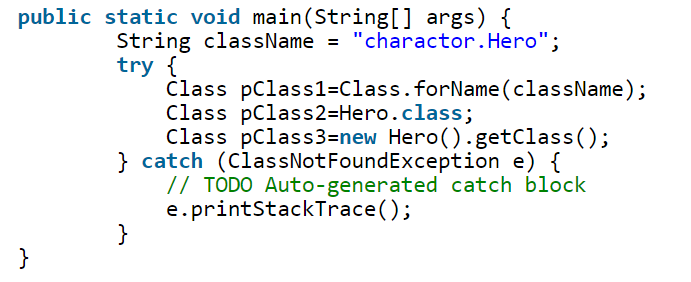
3. new Hero().getClass(); 这种方式没怎么见过，但也很明显

而且在jvm中，一个类只能以一个类对象存在，所以上面三种方式取得的类对象是同一个。



2.获取这个类对象的时候，会同时激活这个类对象的所有类属性(static属性。)





3.对于static类函数，即绑定在类对象上的函数，如果使用安全锁的话，同步对象就是这个类对象。

原文:

当synchronized修饰静态方法的时候， 同步对象就是这个类的类对象。

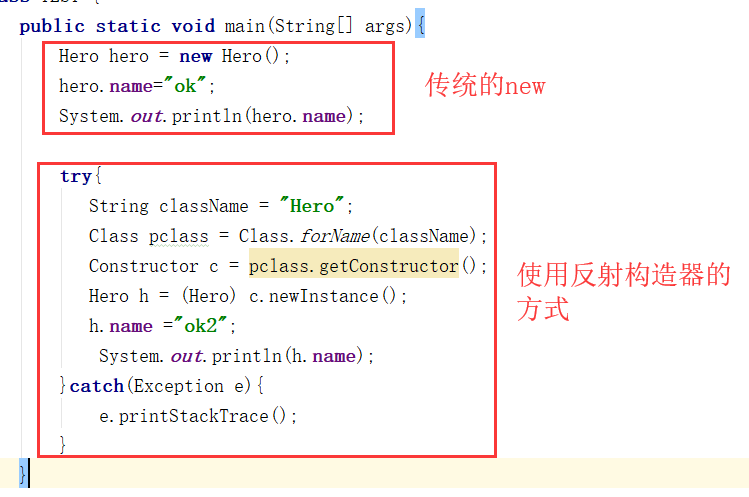


不使用安全锁，20个线程几乎同时启动，并休眠5S

使用安全锁，每个线程就得排队等待5S获得TEST类对象，在进行操作。

## 创建对象：

与传统的通过new 来获取对象的方式不同  
反射机制，会先拿到Hero的“类对象”,然后通过类对象获取“构造器对象”  
再通过构造器对象创建一个对象



下面是关键

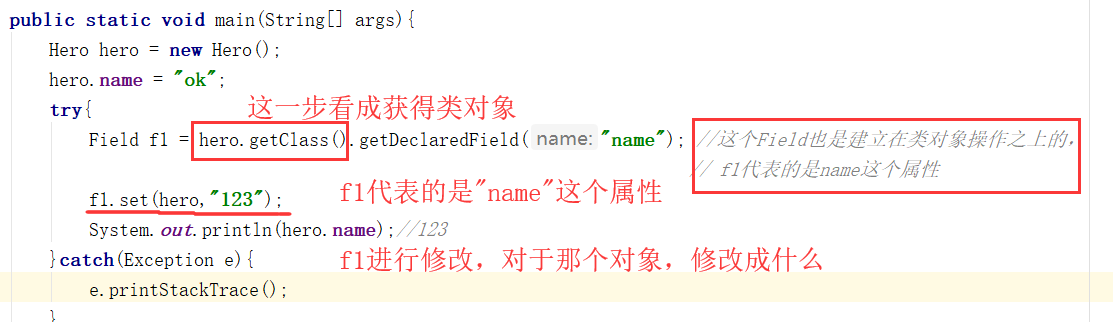
这种编程的思想叫做基于配置的编程思想，编程人员可以通过文件配置来控制生产怎样的对象，配置各种参数控制不同的属性。

Spring等很多框架都是使用的这种配置文件的编程思想。



## 通过类对象修改对象属性 / 调用方法

通过GetDeclaredField获取类对象的字段。

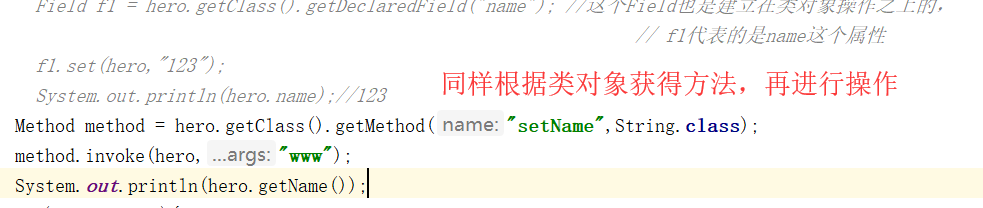


GetDeclaredField就是反射机制。反射出类对象的属性。

原文：

getField和getDeclaredField的区别  
这两个方法都是用于获取字段  
getField **只能获取**public的，包括**从父类继承**来的字段。  
getDeclaredField 可以获取本类所有的字段，**包括private**的，但是**不能获取继承**来的字段。 (**注**： 这里只能获取到private的**字段**，但并不能访问该private字段的**值**,除非加上**setAccessible(true)**)

2.调用方法：



谁调用，参数是什么

## 4.综合练习/总结：(基于类对象)



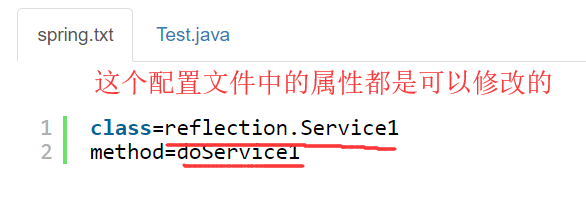
总结就是，反射机制是基于 “类对象”的操作，用类对象结合外界输入的参数去控制生产不同的对象。

一定要注意是基于类对象的操作，就算用实例对象，也是用来获取其类对象的。xxx.getClass()

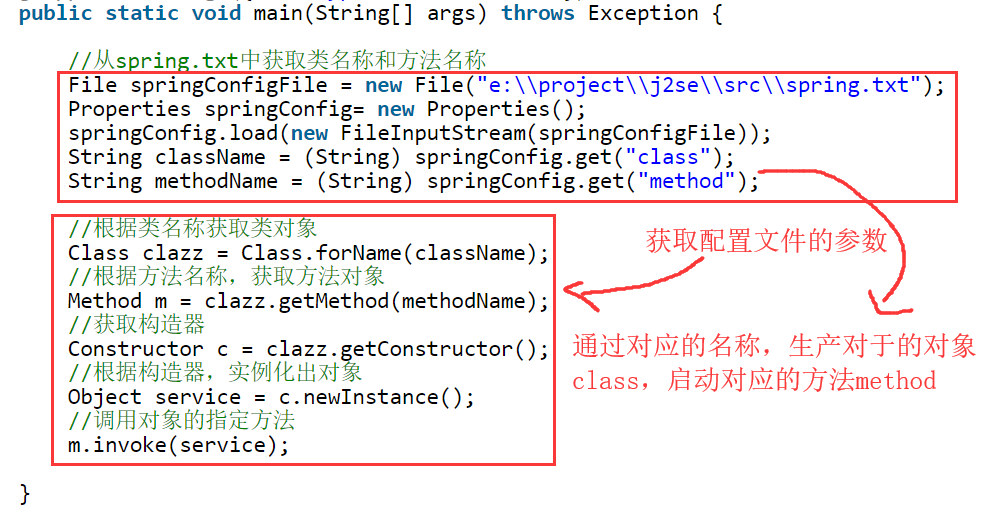
反射机制的作用呢，其实就是基于配置的编程，通过修改配置来获得不同对象，调用不同方法的一种思想。

这种思想非常契合使用框架的思想，使用框架无非就是修改配置来兼容获得不同的功能，而使用反射就能够实现从文件配置—>功能变化的实现。举个例子：

配置文件：



功能实现：



# 注解（类 —> 配置文件）

## 0.注解的实质：

注解实质上也就是注解，他只能用来存在简单的信息（String，int。。。），但使用注解后，整个被注解修饰的类就变成了一个“配置文件”。注解提供的也是配置文件的信息。

之后那些框架（hibernate，spring）的解析类，也是通过这些通过注解方式形成的 ”配置文件” 类（比如Hero），来进行解析注解得到注解的信息，然后通过这些信息和其他框架（如数据库）的整合

所以实质就是，注解，将这个类，变成了一个“配置文件”。

## 自定义注解：

首先有内置注解5个，链接：

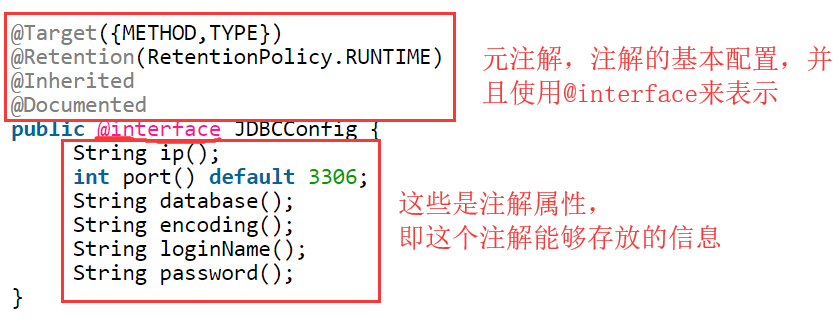
[@Override](https://how2j.cn/k/annotation/annotation-system/1060.html#step4028)  
[@Deprecated](https://how2j.cn/k/annotation/annotation-system/1060.html#step4029)  
[@SuppressWarnings](https://how2j.cn/k/annotation/annotation-system/1060.html#step4030)  
[@SafeVarargs](https://how2j.cn/k/annotation/annotation-system/1060.html#step4031)  
[@FunctionalInterface](https://how2j.cn/k/annotation/annotation-system/1060.html#step4032)

5个基本注解的用法都在对应连接当中。

自定义注解：

1. 注解也是一种类，只不过这种类比较特殊，他可以看成是一种插件类，装配在类对象上面的插件。这是一种新的类，或者说小型属性容器

这是一个最简单的自定义注解和使用方法，关于元注解下面会有解释。



使用方法：



## 元注解的解释：

元注解是对自定义注解的最基本配置，想当于这个注解的配置信息。

### 1.[@Target](mailto:1.@Target)

@Target表示这个注解能够放在什么位置上，（类，方法，属性）

上面的那个注解的taget就是能够放在方法METHOD和TYPE类（接口）上。

一共有下面几个属性可供选择：

TYPE：能修饰类、接口或枚举类型  
FIELD：能修饰成员变量  
METHOD：能修饰方法  
PARAMETER：能修饰参数  
CONSTRUCTOR：能修饰构造器  
LOCAL\_VARIABLE：能修饰局部变量  
ANNOTATION\_TYPE：能修饰注解  
PACKAGE：能修饰包

### 2.[@Retention](mailto:1.@Target)

@Retention是生命周期的意思，表示这个注解能够在程序的那个阶段起作用。

**RetentionPolicy.SOURCE：** 注解只在源代码中存在，编译成class之后，就没了。[@Override](https://how2j.cn/k/annotation/annotation-system/1060.html" \l "step4028)就是这种注解。  
**RetentionPolicy.CLASS：** 注解在java文件编程成.class文件后，依然存在，但是运行起来后就没了。@Retention的默认值，即当没有显式指定@Retention的时候，就会是这种类型。  
**RetentionPolicy.RUNTIME：** 注解在运行起来之后依然存在，程序可以通过反射获取这些信息，[自定义注解@JDBCConfig](https://how2j.cn/k/annotation/annotation-customize/1056.html#step4035) 就是这样。

**一般自定义使用的都是第三种RUNTIME，因为前两种在程序运行的过程中根本不起作用了。**

### 3.[@Inherit](mailto:1.@Target)

继承。顾名思义。这个表示父类的注解也能够被子类继承。

比如有个DBUtil的子类DBUtilChild，那么在这个子类当中也可以使用父类注解中的信息。

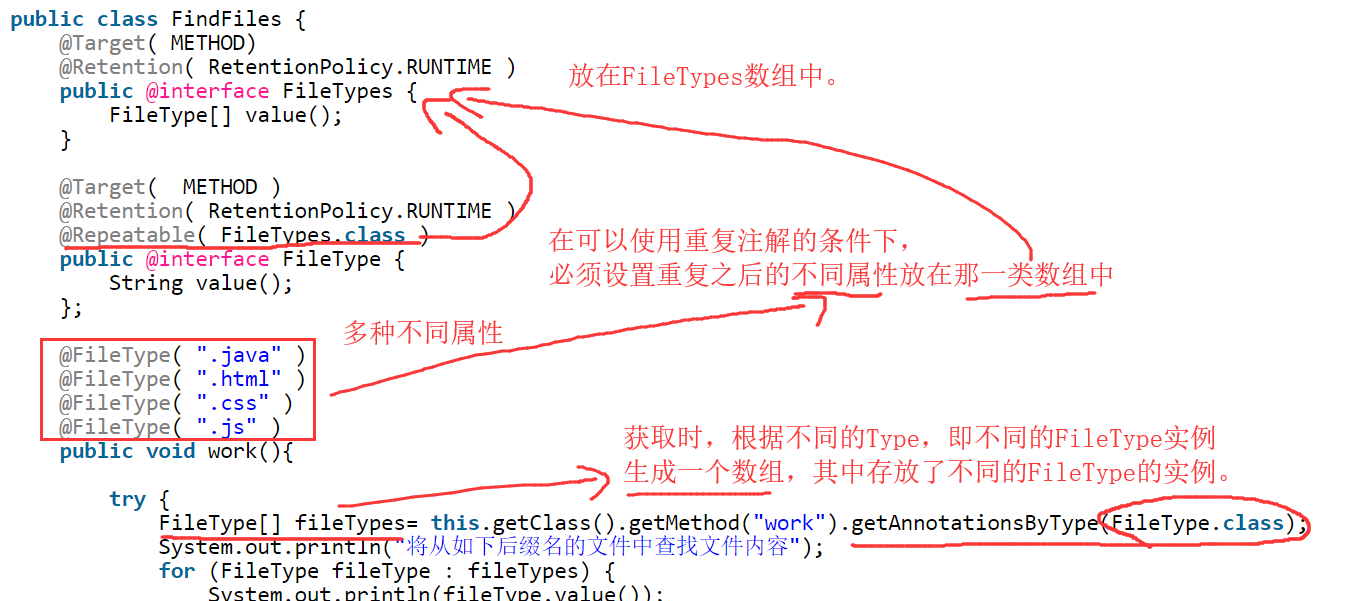


### 4.[@Document](mailto:1.@Target)

在API中生成自定义注解声明的注解：

@Documented 如图所示， 在用javadoc命令生成API文档后，DBUtil的文档里会出现该注解说明。  
  
**注：** 使用eclipse把项目中的.java文件导成API文档步骤：  
1. 选中项目  
2. 点开菜单File  
3. 点击Export  
4. 点开java->javadoc->点next  
5. 点finish

### 5.@Repeatable：



多种FileType（5个）🡪放在FileTypes数组中 –>取出时根据Type(不同实例)，生成一个FileType数组,

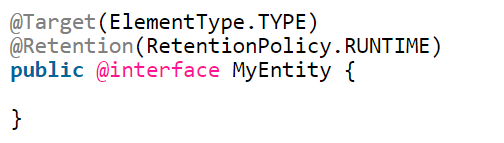
这里的FileTypes临时存放数组貌似是个工具人，在取出时好像都没涉及到。

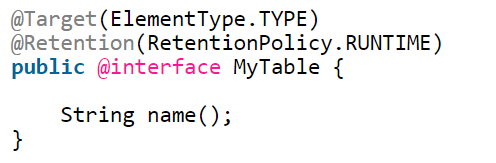
## 模仿hibernate

Hibernate的注解常用的五种：Id，GeneratedValue，Table，Entity，Column

这里模仿就是前面加个My标注。

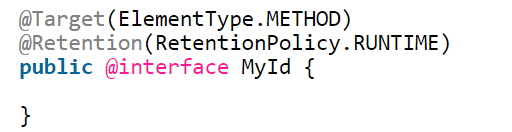
### 1.设计5种注解：

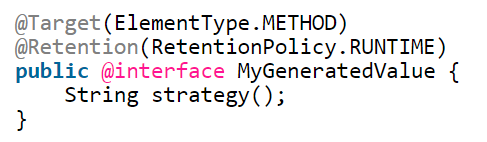


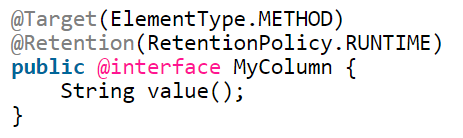
.

这两种是放在类上面作用的。

下面的注解是针对属性的，放在方法METHOD上面

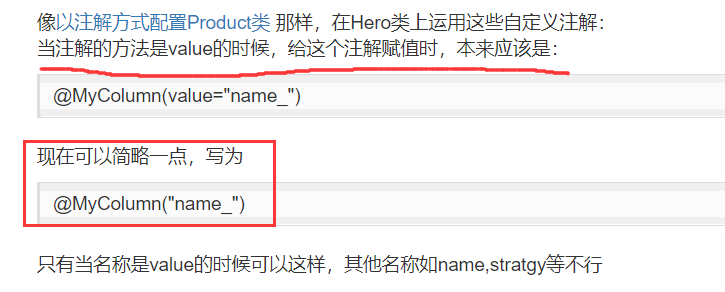






### 2.作用于Hero类

注意一点：如果属性是以value()为装载的，那么在给其注解赋值的时候可以不用写value=”xxx”，直接在括号内写内容即可，但如果是其他的比如name(),strategy()装载的话，就要写全。



然后就这样了:



### 3.设计解析类（获取注解内容，框架核心）

解析类ParseHibernateAnnotation，用来通过获取Hero类对象上的注解来得到注解中的信息，进行整合数据库操作。

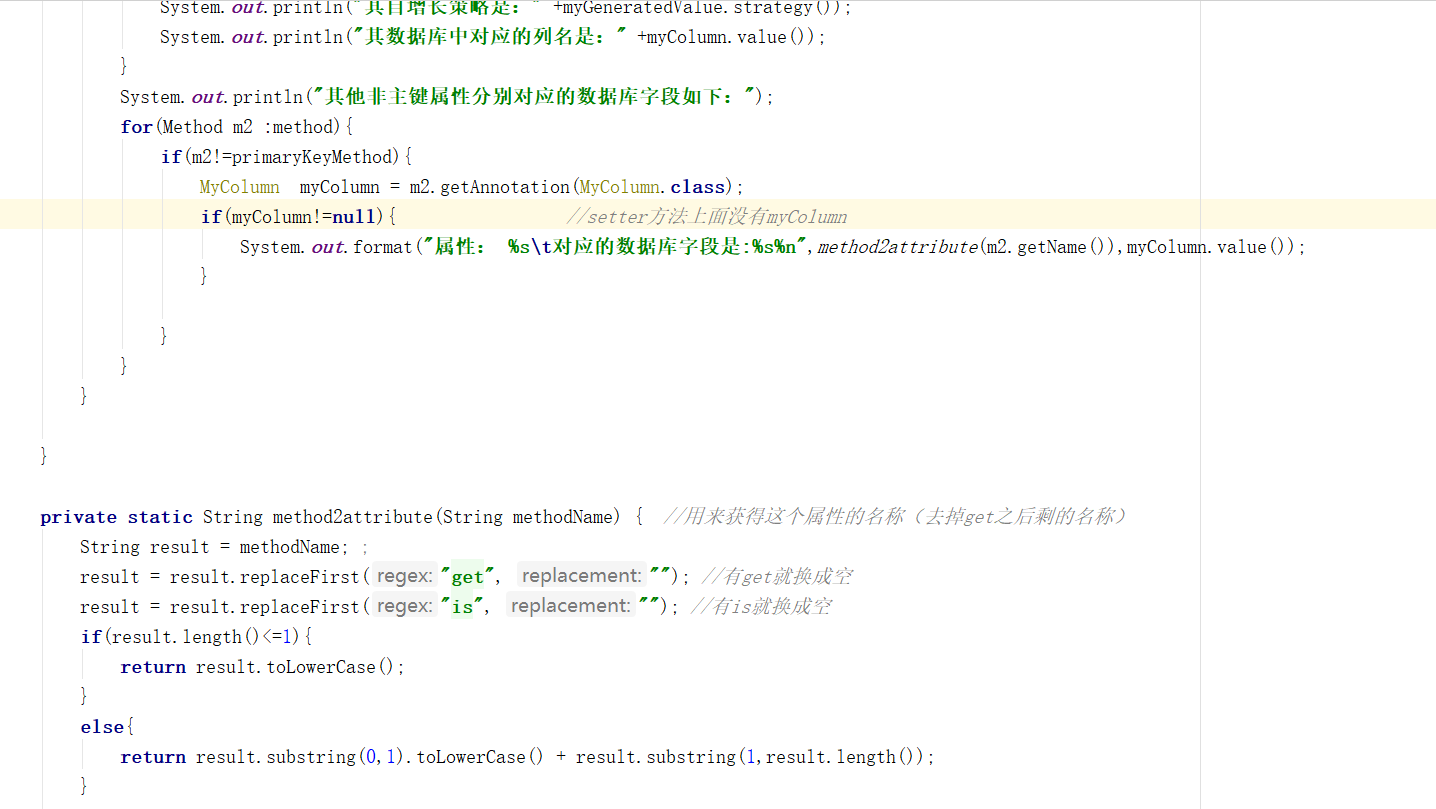
这个类也就是hibernate(或者其他使用注解的框架的核心解释类，通过类对象上的注解来得到相应信息来进行操作)

到这里也就恍然大悟，注解实质上也就是注解，使用注解后，整个被注解修饰的类就变成了一个配置文件。而框架的解析类，也是通过这些通过注解方式形成的 ”配置文件” 类（比如Hero），来进行解析注解得到信息，然后通过这些信息和其他框架（如数据库）的整合

步骤如下：

1. 首先获取Hero.class类对象  
   2. 判断本类是否进行了MyEntity 注解  
   3. 获取注解 MyTable  
   4. 遍历所有的方法，如果某个方法有MyId注解，那么就记录为主键方法primaryKeyMethod  
   5. 把主键方法的自增长策略注解MyGeneratedValue和对应的字段注解MyColumn 取出来，并打印  
   6. 遍历所有非主键方法，并且有MyColumn注解的方法，打印属性名称和字段名称的对应关系。





总的来说就是通过获取类对象Hero和解析其注解内容的过程。

### 注解分类：

简单的将注解分个类：

1.作用域分类：

理解为

1.编译为.class之前，只能源码存在的注解，如override

2.编译为.class之后但运行时消失了的

3.程序运行时依旧生效的（这种才是核心注解）

原文：

根据注解的作用域[@Retention](https://how2j.cn/k/annotation/annotation-meta/1057.html" \l "step4039)，注解分为  
RetentionPolicy.SOURCE： Java源文件上的注解  
RetentionPolicy.CLASS： Class类文件上的注解  
RetentionPolicy.RUNTIME： 运行时的注解

2.来源区分：

可以分为三类：

1.java自带注解override一类

2.自定义注解

3.第三方框架注解：hibernate，spring (其实也就是自定义注解)

原文：

在工作中，大部分都是使用第三方注解， 当然第三方注解本身就是自定义注解。 本教程的主要作用是帮助大家理解这些第三方注解是如何工作的，让大家用得心里踏实一些。