java高级之反射

一、反射入门

在方法区存在这么一些对象，叫做**类对象**，他们表述了我们写的所有的类，当我们new对象时会根据这些类对象，并调用其构造方法为我们创建实例。

JAVA反射机制是在**运行状态中**，对于任意一个类，都能够**知道这个类的所有属性和方法**；对于任意一个对象，都能够调用它的任意一个方法和属性；这种动态获取的信息以及动态调用对象的方法的功能称为java语言的反射机制。

简单的说:一个类有多个组成部分，例如：成员变量，方法，构造方法等。反射就是加载类，并解剖出类的各个组成部分。

Java反射机制主要提供了以下功能：

- 在运行时判断任意一个对象所属的类；
- 在运行时构造任意一个类的对象；
- 在运行时判断任意一个类所具有的成员变量和方法；
- 在运行时调用任意一个对象的方法；生成动态代理

二、反射的API介绍

简述：我们写的所有的类，都会被appclassloader加载到内存的方法区，生成一个Class类型的对象，他们那是你写的class，但同时他们也是Class的实例。也叫说明书的说明书。

Class叫说明书的说明书，我告诉了我们说明书该怎么写，比如可以有方法、属性等等。

我们的class都是说明书，说明了某一个事物有哪些方法和属性。

Java反射所需要的类并不多，主要有java.lang.Class类和java.lang.reflect包中的Field、Constructor、Method、Annotation类。
注意：Class类是Java反射的起源，针对任何一个你想探勘的类，只有先为它产生一个Class类的对象，接下来才能通过Class对象获取其2

1、获取类对象的方法

```
1、使用类
Class clazz = Dog.class;

2、使用全类名
Class aClass = Class.forName("com.xinzhi.Day");

3、使用对象
Dog dog = new Dog();
Class clazz = dog.getClass();
```

2、对类对象操作

```
//获取类名字
String name = clazz.getName();
//获取类加载器
ClassLoader classLoader = clazz.getClassLoader();
//获取资源
```

```
URL resource = clazz.getResource("");
//得到父类
Class superclass = clazz.getSuperclass();
//判断一个类是不是接口，数组等等
boolean array = clazz.isArray();
boolean anInterface = clazz.isInterface();

//重点，使用class对象实例化一个对象
Object instance = clazz.newInstance();
```

3、获取字段并操作

> 字段 Field

（1）获取字段

```
//获取字段，只能获取公共的字段（public）
Field name = clazz.getField("type");
Field[] fields = clazz.getFields();
//能获取所有的字段包括private
Field color = clazz.getDeclaredField("color");
Field[] fields = clazz.getDeclaredFields();

System.out.println(color.getType());
```

（2）获取对象的属性

```
Dog dog = new Dog();
dog.setColor("red");
Class clazz = Dog.class;
Field color = clazz.getDeclaredField("color");
System.out.println(color.get(dog));
```

当然你要是明确类型你还能用以下方法，

```
Int i = age.getInt(dog);
xxx.getDouble(dog);
xxx.getFloat(dog);
xxx.getBoolean(dog);
xxx.getChar(dog);
//每一种基本类型都有对应方法
```

（3）设置对象的属性

```
Dog dog = new Dog();
dog.setColor("red");
Class clazz = Dog.class;
Field color = clazz.getDeclaredField("color");
color.set(dog,"blue");
System.out.println(dog.getColor());
```

当然如果你知道对应的类型，我们可以这样

```
xxx.setBoolean(dog,true);
xxx.getDouble(dog, 1.2);
xxx.getFloat(dog,1.2F);
xxx.getChar(dog,'A');
//每一种基本类型包装类都有对应方法
```

```
Field color = dogClass.getDeclaredField("color");
//暴力注入
color.setAccessible(true);
color.set(dog,"red");
```

3、方法

（1）获取方法

```
//根据名字和参数类型获取一个方法
Method method = clazz.getMethod("eat",String.class);
Method[] methods = clazz.getMethods();

Method eat = clazz.getDeclaredMethod("eat", String.class);
Method[] declaredMethods = clazz.getDeclaredMethods();
```

（2）对方法的操作

```
Dog dog = new Dog();
dog.setColor("red");
Class clazz = Dog.class;
//获取某个方法，名字，后边是参数类型
Method method = clazz.getMethod("eat",String.class);
//拿到参数的个数
int parameterCount = method.getParameterCount();
//拿到方法的名字
String name = method.getName();
//拿到参数的类型数组
Class<?>[] parameterTypes = method.getParameterTypes();
//拿到返回值类型
Class<?> returnType = method.getReturnType();
//重点。反射调用方法，传一个实例，和参数
method.invoke(dog,"热狗");
```

```
        Class dogClass = Class.forName("com.xinzhi.Dog");
        Object dog = dogClass.newInstance();

        Method eat = dogClass.getMethod("eat");
        eat.invoke(dog);

        Method eat2 = dogClass.getMethod("eat",String.class);
        eat2.invoke(dog,"meat");

        Method eat3 = dogClass.getMethod("eat",String.class,int.class);
        eat3.invoke(dog,"meat",12);
```

4.构造函数

（1）获取并构建对象

```
Constructor[] constructors = clazz.getConstructors();
Constructor constructor = clazz.getConstructor();
Constructor[] declaredConstructors = clazz.getDeclaredConstructors();
Constructor declaredConstructor = clazz.getDeclaredConstructor();


Object obj = constructor.newInstance();
```

5、注解

（1）从方法、字段、类上获取注解

```
//元注解 要加上runtime
//类上
Annotation annotation = clazz.getAnnotation(Bean.class);
Annotation[] annotations = clazz.getAnnotations();

//字段上
Annotation annotation = field.getAnnotation(Bean.class);
Annotation[] annotations = field.getAnnotations();

//方法上
Annotation annotation = method.getAnnotation(Bean.class);
Annotation[] annotations = method.getAnnotations();
```