1. 反射的概念：

主要是指程序可以访问，检测和修改它本身状态或行为的一种能力，并能根据自身行为的状态和结果，调整或修改应用所描述行为的状态和相关的语义。

反射是 **Java** 中一种强大的工具，能够使我们很方便的创建灵活的代码，这些代码可以再运行时装配，无需在组件之间进行源代码链接。但是反射使用不当会成本很高！

2.反射机制

A.是什么

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。

B. 作用：

A.反编译：.class-->.java

B.通过反射机制访问 java 对象的属性，方法，构造方法等；

这样我们就**可以获得类的各种内容，进行了反编译**。对于 JAVA 这种先编译再运行的语言来说，反射机制可以使代码更加灵活，更加容易实现面向对象。

反射在 Java 的框架中使用的最为普遍。

C.能做什么？ 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。

3. 反射机制中的常用类：5 个

java.lang.Class; ----表示正在运行的 Java 应用程序中的类和接口

java.lang.reflect.Constructor; ----表示构造函数

java.lang.reflect.Field; ----表示字段

java.lang.reflect.Method; ----表示方法

java.lang.reflect.Modifier; -----表示修饰符

4. 获取类有三种方法：

第一种方式： **使用 Class 的 forName 方法,**例如：

```
Class
c1 = Class.forName("com.ddb.javacore.reflect.Person");
```

第二种方式： **类名.class** ，例如：

```
//java 中每个类型都有 class 属性.
Class c2 = Person.class;
```

第三种方式： **对象名.getClass()** 例如：

```
//java 语言中任何一个 java 对象都有 getClass 方法
Person person = new Person();
Class c3 = person.getClass(); //c3 是运行时类 (e 的运行时类是 Person)
```

5. 创建对象：

A. 获取类以后我们来创建它的对象，利用 newInstance(). 例如：

```
Class c =Class.forName("com.ddb.javacore.reflect.Person");
//创建此 Class 对象所表示的类的一个新实例
Object o = c.newInstance(); //调用了 Employee 的无参数构造方法.
```

B. 使用构造函数，例如：

```
// 取得全部的构造函数
Constructor<?> cons[] = clazz.getConstructors();
per1 = (Person) cons[3].newInstance();
per3 = (Person) cons[1].newInstance(20);
per4 = (Person) cons[0].newInstance("Rollen", 20);
```

6. 获取属性：分为所有的属性和指定的属性：

a，先看获取所有的属性的写法：

```
Field[] fs = c.getDeclaredFields();
```

b，获取特定的属性：

```
Field idF = c.getDeclaredField("id");//获取 id 属性
```

利用反射访问字段有几种方式？2 种：

A. 获取它所有的属性 对象.getDeclarFields（）

B. 通过指定的方法名或属性名获取到指定的字段或方法对象.getDeclarField（）

到现在为止，我们访问对象属性的方法有：

A. 常规的 getter 和 setter 方法；

B. 利用反射的暴力访问；

C. 利用反射的温柔访问---通过反射得到访问属性的 getter 和 setter 方法访问！

7. 获取方法和构造方法

方法关键字	含义
getDeclaredMethods()	获取所有的方法

getReturnType()	获得方法的放回类型
getParameterTypes()	获得方法的传入参数类型
getDeclaredMethod("方法名",参数类型.class,……)	获得特定的方法
构造方法关键字	含义
getDeclaredConstructors()	获取所有的构造方法
getDeclaredConstructor(参数类型.class,……)	获取特定的构造方法
父类和父接口	含义
getSuperclass()	获取某类的父类
getInterfaces()	获取某类实现的接口

如何通过反射创建被反射对象的一个实例；

A.通过 clazz 对象 ----clazz. newInstance(); //只能针对无参构造方法

B.通过 construtor 对象 ---- constructor. newInstance(); //最通用的方式，支持有参和无参构

8.使用反射设定属性值的步骤：

A.得到被反射对象的 clazz 对象， 例如：Class<?> clazz = Person.class;

B.得到要修改的属性值， 例如：Field fname = class1.getDeclaredField("name");

C.打开属性的可访问性， 例如：fname.setAccessible(true);

D.创建被反射对象的实例， 例如：Person obj = (Person) clazz.newInstance();

E.设定属性值， 例如：fname.set(obj, "zhangsan01");

F.验证属性值被修改。

9.使用反射唤醒方法的步骤：

主要**通过 method.invoke()来唤醒方法，参数 1---被反射对象的实例，[可选]参数 2….**

A.得到被反射对象的 clazz 对象， 例如：Class<?> clazz = Person.class;

B.得到要唤醒的方法， 例如： method = clazz.getMethod("reflect2", int.class, String.class);

C.创建被反射对象的实例， 例如：obj=clazz. newInstance()；或者 obj=constructor. newInstance()；

D.唤醒(执行)指定的方法， 例如： method.invoke(obj, 20, "张三");

---method.invoke(obj,args);//obj - 从中调用底层方法的对象，args - 用于方法调用的参数

E.确认方法被执行！

10.注意：

在用反射设定值和唤醒方法之前，最好确认是否存在。

如果不知道是否存在，获取到所有的属性和方法判断后再使用。

练习题：

利用反射，获取 String 类的属性和方法在打印输出到文本文件中！！

其他知识：

扫描 PDF 的高亮方法：

A.安装 Adobe Acrobat Pro DC；

B.使用该软件打开扫描的 PDF 文件；

C.点击左侧三角，展开左侧菜单栏，单击增强扫描；

D.在工具栏的下方会出现一排增强扫描的工具栏，选择文本识别--在本文件中；

E.设置语言----识别文本----保存；

F.添加高亮或注释！！！

参考链接：

JAVA 中的反射机制 - 学会改变自己——才能突破 - 博客频道 - CSDN.NET
http://blog.csdn.net/liujiahan629629/article/details/18013523

Java 反射机制详解 - Java 初级码农 - 博客园
http://www.cnblogs.com/lzq198754/p/5780331.html

附属代码：

获取所有属性的代码：

```
1. //获取整个类
2.             Class c = Class.forName("java.lang.Integer");
3.               //获取所有的属性?
4.             Field[] fs = c.getDeclaredFields();
5.               //定义可变长的字符串，用来存储属性
```

```
6.              StringBuffer sb = new StringBuffer();
7.              //通过追加的方法，将每个属性拼接到此字符串
   中
8.              //最外边的 public 定义
            sb.append(Modifier.toString(c.getModifiers())
 + " class " + c.getSimpleName() +"{\n");
9.              //里边的每一个属性
10.             for(Field field:fs){
11.                 sb.append("\t");//空格
12.                 sb.append(Modifier.toString(fiel
   d.getModifiers())+" ");//获得属性的修饰符，例如
   public，static 等等
13.                 sb.append(field.getType().getSim
   pleName() + " ");//属性的类型的名字
14.                 sb.append(field.getName()+";\n")
   ;//属性的名字+回车
15.             }
16.             sb.append("}");
17.             System.out.println(sb);
```

获取并设置指定属性的代码：

```
1. public static void main(String[] args) throws Except
   ion{
2.     /*
3.     User u = new User();
4.     u.age = 12; //set
5.     System.out.println(u.age); //get
6.     */
7.     //获取类
8.     Class c = Class.forName("User");
```

```
9.      //获取 id 属性
10.     Field idF = c.getDeclaredField("id");
11.     //实例化这个类赋给 o
12.     Object o = c.newInstance();
13.     //打破封装
14.     idF.setAccessible(true); //使用反射机制可以打破封装
        性，导致了 java 对象的属性不安全。
15.     //给 o 对象的 id 属性赋值"110"
16.     idF.set(o, "110"); //set
17.     //get
18.     System.out.println(idF.get(o));
19. }
```

唤醒方法的案例代码：

```
package net.xsoftlab.baike;
    import java.lang.reflect.Method;
    public class TestReflect {
    public static void main(String[] args) throws Exception {
    Class<?> clazz = Class.forName("net.xsoftlab.baike.TestReflect");
    // 调用 TestReflect 类中的 reflect1 方法
    Method method = clazz.getMethod("reflect1");
    method.invoke(clazz.newInstance());
    // Java 反射机制 - 调用某个类的方法 1.
    // 调用 TestReflect 的 reflect2 方法
    method = clazz.getMethod("reflect2", int.class, String.class);
    method.invoke(clazz.newInstance(), 20, "张三");
    // Java 反射机制 - 调用某个类的方法 2.
    // age -> 20. name -> 张三
    }
    public void reflect1() {
    System.out.println("Java 反射机制 - 调用某个类的方法 1.");
    }
    public void reflect2(int age, String name) {
    System.out.println("Java 反射机制 - 调用某个类的方法 2.");
    System.out.println("age -> " + age + ". name -> " + name);
    }
    }
```