# 1.反射

## 1.1什么是反射

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。

要想解剖一个类,必须先要获取到该类的字节码文件对象。而解剖使用的就是Class类中的方法.所以先要获取到每一个字节码文件对应的Class类型的对象.  
例如：一个类有：成员变量、方法、构造方法、包等等信息，利用反射技术可以对一个类进行解剖，把个个组成部分映射成一个个对象。

## 1.2反射使用场景

耦合---模块之间产生了紧密的依赖的时候，产生了耦合

解耦---降低模块之间的依赖性

数据库：常见数据库：Oracle，MySQL，SQLServer（java中用的偏少，微软研发），DB2（IBM研发，银行领域用的偏多），SQLLite（用于移动端），

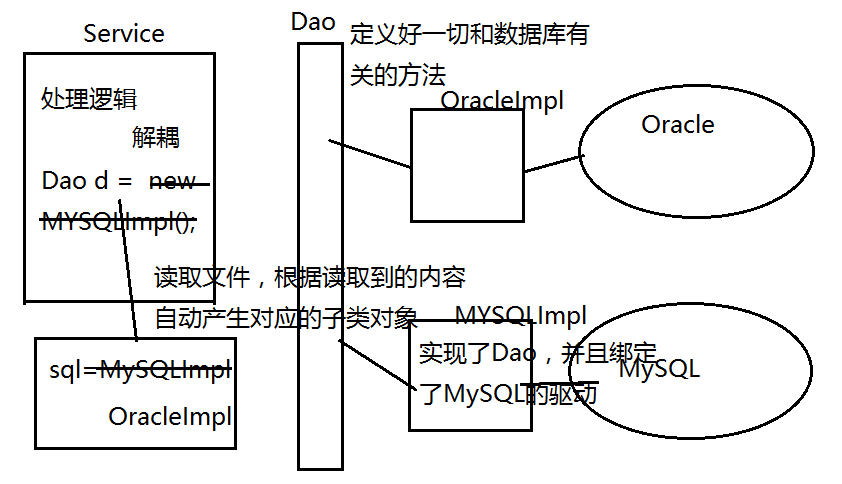
MySQL：目前是免费的，提供的是数据库最基本的操作，存储的数据量大概是千万级。

Oracle：收费的，存储的数据量达到上亿，售后服务良好。

集群（Cluster）：集群式一组相互独立的、通过高速网络互连的计算机，它们构成了一组，并以单一系统的模式加以管理。

耦合（Coupling）：模块之间绑定。程序结构中各个模块之间相互关联的度量。

解耦（Decoupling）：降低两个模块之间的依赖性。



## 1.3反射的作用

反射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。

反射提供封装程序集、模块和类型对象。

反射其实是解剖类，分析类的字节码。产生类所对应的实例对象。

实现解耦操作，便于模块化开发。

## 1.4解剖类

1.4.1 Class

代表字节码的类，代表类的类。每一个Class对象都对应了一个类/接口。

1.4.2 class重要方法

forName(String className);

获取指定的字节码对象。

forName(String name,boolean initialize,ClassLoader loader);

getConstructor(Class<?>...parameterTypes);

获取指定的公有构造函数。

Class<String> clz = String.class;

String(byte[],int,int)

// 只能获取public的构造函数

Constructor<String> c = clz.getConstructor(byte[].class, int.class,int.class);

String str = c.newInstance(new byte[] { 97, 98, 99, 100, 101, 102,103 }, 1, 3);

System.out.println(str);

getConstructors();

获取所有公有构造函数。返回的是一个构造函数数组。

// 表示获取String类中所有public的构造函数

Constructor[] cs = clz.getConstructors();

for (Constructor constructor : cs) {

System.out.println(constructor);

}

getDeclaredConstructor(Class<?>...parameterTypes);

获取指定的构造函数。

// 表示获取指定的形式的构造函数

Constructor<String> c = clz.getDeclaredConstructor(char[].class, boolean.class);

// 暴力破解/暴力拆除

c.setAccessible(true);

String str = c.newInstance(new char[] { 'a', 'b', 'c' }, true);

System.out.println(str);

getDeclaredConstructors();

获取所有构造函数。返回的是一个构造函数数组。

getDeclaredField(String name);

获取指定属性。

getDeclaredFields();

获取所有属性，返回一个数组。

getDeclaredMethod(String name,Class<?>...parameterTypes);

获取实例对象的方法，返回的是一个方法对象。

Class<String> clz = String.class;

String str = "abcdefg";

// 获取getBytes()

// byte[] bs = str.getBytes();

Method m = clz.getDeclaredMethod("getBytes");

// 执行方法对象

// 第一个参数表示方法作用的对象

// 第二个参数表示方法执行需要的实际参数

byte[] bs = (byte[]) m.invoke(str);

System.out.println(bs);

getDeclaredMethods();

获取所有方法。返回一个数组。

getField(String name);

获取公有属性。

getFields();

获取所有公有属性。

getInterfaces();

表示获取这个类或者接口实现的所有接口。返回的是一个数组类型。

Class<String> clz = String.class;

// 获取这个类/接口实现的所有接口

Class[] cs = clz.getInterfaces();

for (Class class1 : cs) {

System.out.println(class1);

}

getSuperclass();

获取父类。

getName();

获取全路径名称。

getSimpleName();

获取类名简称。

getPackage();

获取类所在包。

isAnonymousClass();

判断是否是匿名内部类。

isAssignableFrom();

判断参数是否是字节码对象的子类。

isInstance();

判断参数是否是字节码对象以及子类的实例。

isLocalClass();

判断是否是方法内部类。

isMemberClass();

判断是否是成员内部类。

isArray();

判断是否是数组。

isEnum();

判断是否是枚举。

isInterface();

判断是否是接口。

isPrimitive();

判断是否是基本类型。

newInstance();

创建一个字节码对象的实例。

1.4.3 Field

代表属性的类

1.4.4 Field重要方法

set(Object o,Object value);

设置属性值。

get(Object o);

获取属性值。

getType();

获取属性类型。

public class ClassDemo6 {

@SuppressWarnings("rawtypes")

public static void main(String[] args) throws Exception {

Class<String> clz = String.class;

String str = "abc";

// 获取指定的属性

// str.setHash(123);

Field f = clz.getDeclaredField("hash");

f.setAccessible(true);

// 设置属性值

f.set(str, 123);

// 获取属性值

System.out.println(f.get(str));

// 获取属性的声明类型

Class c = f.getType();

System.out.println(c);

}

}

1.4.5 Method

代表方法的类

1.4.4Constructor

代表构造方法的类

1.4.5Package

代表包的类

1.4.6Annotation

代表注解的类

## 1.5 Class类

在java中任何一个类都有一个类的class类，获取Class对象的方法：

通过类名.class来获取一个字节码对象

通过对象的getClass()来获取一个字节码对象

通过Class.forName()来获取一个字节码对象，要求传入的是类的全路径名

## 1.4设计模式

1.4.1 装饰者模式

装饰模式指的是在不必改变原类文件和使用继承的情况下，动态地扩展一个对象的功能。它是通过创建一个包装对象，也就是装饰来包裹真实的对象。

BufferedReader(Reader in)---底层读取数据靠的传入的作为参数的流对象，BufferedReader在源流的基础上给增加了一个缓冲区

继承的缺点：

1. 单继承

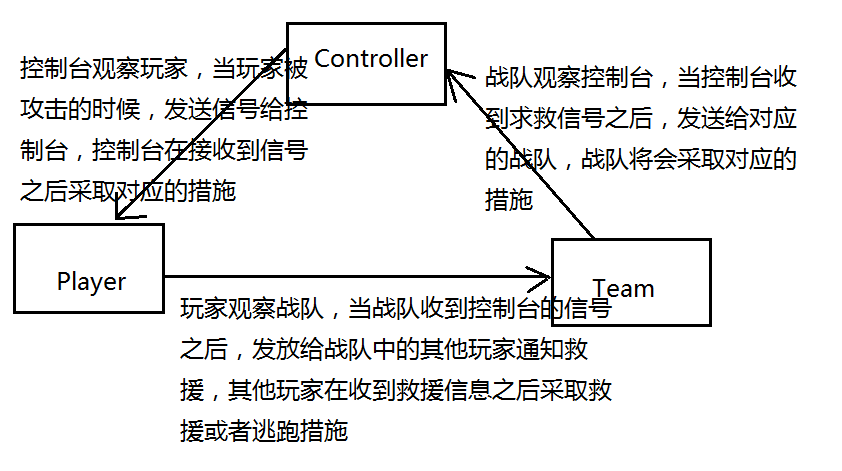
2. 对于已经存在的父类对象，无法使之使用子类中重写后的方法

3. 对于子类对象而言，无法再次调用父类中的原来的没改变过方法

利用一个统一的父类或者接口，将一个子类对象作为参数来构造另一个子类对象，对其中的方法做改变或者增强---装饰模式

1.4.2观察者模式

将一个对象作为观察目标，当观察目标发生改变的时候，观察者在检测到改变信号之后采取相应的行为应对变化的这个过程---观察者模式



# 2.JDK8的部分特性

## 2.1接口中的默认方法

从JDK1.8开始，接口中允许定义实体方法，但是必须使用default修饰，从JDK1.8开始，在接口中也允许定义静态方法

## 2.2Lambda表达式

(parameters) -> expression;

(parameters) -> {statements;}

Lambda表达式实际上是在重写一个抽象方法---只能作用于接口，并且要求这个接口中只能有一个抽象方法---函数式接口