**[面试题思考：什么是 Java 的反射机制](https://www.cnblogs.com/songanwei/p/9386749.html)**

**一、反射机制概述**

Java 反射机制是在运行状态中，对于任意一个类，都能够获得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属性和方法。这种在运行时动态的获取信息以及动态调用对象的方法的功能称为Java 的反射机制。

Class 类与java.lang.reflect 类库一起对反射的概念进行了支持，该类库包含了Field,Method,Constructor类(每个类都实现了Member 接口)。这些类型的对象时由JVM 在运行时创建的，用以表示未知类里对应的成员。

这样你就可以使用Constructor 创建新的对象，用get() 和set() 方法读取和修改与Field 对象关联的字段，用invoke() 方法调用与Method 对象关联的方法。另外，还可以调用getFields() getMethods() 和 getConstructors() 等很便利的方法，以返回表示字段，方法，以及构造器的对象的数组。这样匿名对象的信息就能在运行时被完全确定下来，而在编译时不需要知道任何事情。

**二、获取字节码的方式**

在Java 中可以通过三种方法获取类的字节码(Class)对象

* 通过Object 类中的getClass() 方法，想要用这种方法必须要明确具体的类并且创建该类的对象。
* 所有数据类型都具备一个静态的属性.class 来获取对应的Class 对象。但是还是要明确到类，然后才能调用类中的静态成员。
* 只要通过给定类的字符串名称就可以获取该类的字节码对象，这样做扩展性更强。通过Class.forName() 方法完成，必须要指定类的全限定名，由于前两种方法都是在知道该类的情况下获取该类的字节码对象，因此不会有异常，但是Class.forName() 方法如果写错类的路径会报 ClassNotFoundException 的异常。

按 Ctrl+C 复制代码



按 Ctrl+C 复制代码

**三、通过反射机制获取类信息**

通过反射机制创建对象，在创建对象之前要获得对象的构造函数对象，通过构造函数对象创建对应类的实例。

下面这段代码分别在运行期间创建了一个无参与有参的对象实例。由于getConstructor() 方法与newInstance() 方法抛出了很多异常(你可以通过源代码查看它们)，这里就简写了直接抛出一个Exception，下同。

[复制代码](javascript:void(0);)

package com.jas.reflect;

import java.lang.reflect.Constructor;

public class ReflectTest {

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

Class clazz = null;

clazz = Class.forName("com.jas.reflect.Fruit");

Constructor<Fruit> constructor1 = clazz.getConstructor();

Constructor<Fruit> constructor2 = clazz.getConstructor(String.class);

Fruit fruit1 = constructor1.newInstance();

Fruit fruit2 = constructor2.newInstance("Apple");

}

}

class Fruit{

public Fruit(){

System.out.println("无参构造器Run...........");

}

public Fruit(String type){

System.out.println("有参构造器Run..........." + type);

}

}

[复制代码](javascript:void(0);)

输出：

无参构造器Run………..

有参构造器Run………..Apple

通过反射机制获取Class 中的属性

[复制代码](javascript:void(0);)

package com.jas.reflect;

import java.lang.reflect.Field;

public class ReflectTest {

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

Class<?> clazz = null;

Field field = null;

clazz = Class.forName("com.jas.reflect.Fruit");

//field = clazz.getField("num"); getField() 方法不能获取私有的属性

// field = clazz.getField("type"); 访问私有字段时会报 NoSuchFieldException异常

field = clazz.getDeclaredField("type"); //获取私有type 属性

field.setAccessible(true); //对私有字段的访问取消检查

Fruit fruit = (Fruit) clazz.newInstance(); //创建无参对象实例

field.set(fruit,"Apple"); //为无参对象实例属性赋值

Object type = field.get(fruit); //通过fruit 对象获取属性值

System.out.println(type);

}

}

class Fruit{

public int num;

private String type;

public Fruit(){

System.out.println("无参构造器Run...........");

}

public Fruit(String type){

System.out.println("有参构造器Run..........." + type);

}

}

[复制代码](javascript:void(0);)

输出：

无参构造器Run………..

Apple

通过反射机制获取Class 中的方法并运行。

[复制代码](javascript:void(0);)

package com.jas.reflect;

import java.lang.reflect.Constructor;

import java.lang.reflect.Method;

public class ReflectTest {

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

Class clazz = null;

Method method = null;

clazz = Class.forName("com.jas.reflect.Fruit");

Constructor<Fruit> fruitConstructor = clazz.getConstructor(String.class);

Fruit fruit = fruitConstructor.newInstance("Apple"); //创建有参对象实例

method = clazz.getMethod("show",null); //获取空参数show 方法

method.invoke(fruit,null); //执行无参方法

method = clazz.getMethod("show",int.class); //获取有参show 方法

method.invoke(fruit,20); //执行有参方法

}

}

class Fruit{

private String type;

public Fruit(String type){

this.type = type;

}

public void show(){

System.out.println("Fruit type = " + type);

}

public void show(int num){

System.out.println("Fruit type = " + type + ".....Fruit num = " + num);

}

}

[复制代码](javascript:void(0);)

输出：

Fruit type = Apple

Fruit type = Apple…..Fruit num = 20

**四、反射机制简单应用（使用简单工厂创建对象）**

Class.forName() 生成的结果是在编译时不可知的，因此所有的方法特征签名信息都是在执行时被提取出来的。反射机制能过创建一个在编译期完全未知的对象，并调用该对象的方法。

以下是反射机制与泛型的一个应用，通过一个工厂类创建不同类型的实例。

要创建对象的实例类Apple ：

package com.jas.reflect;

public interface Fruit {}

class Apple implements Fruit{}

加载的配置文件config.properties：

Fruit=com.jas.reflect.Apple

工厂类BasicFactory ：

[复制代码](javascript:void(0);)

package com.jas.reflect;

import java.io.FileReader;

import java.util.Properties;

public class BasicFactory {

private BasicFactory(){}

private static BasicFactory bf = new BasicFactory();

private static Properties pro = null;

static{

pro = new Properties();

try{

//通过类加载器加载配置文件

pro.load(new FileReader(BasicFactory.class.getClassLoader().

getResource("config.properties").getPath()));

}catch (Exception e) {

e.printStackTrace();

}

}

public static BasicFactory getFactory(){

return bf;

}

//使用泛型获得通用的对象

public <T> T newInstance(Class<T> clazz){

String cName = clazz.getSimpleName(); //获得字节码对象的类名

String clmplName = pro.getProperty(cName); //根据字节码对象的类名通过配置文件获得类的全限定名

try{

return (T)Class.forName(clmplName).newInstance(); //根据类的全限定名创建实例对象

}catch (Exception e) {

throw new RuntimeException(e);

}

}

}

[复制代码](javascript:void(0);)

创建对象实例：

[复制代码](javascript:void(0);)

package com.jas.reflect;

public class ReflectTest {

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

Fruit fruit = BasicFactory.getFactory().newInstance(Fruit.class);

System.out.println(fruit);

}

}

[复制代码](javascript:void(0);)

输出

com.jas.reflect.Apple@4554617c

上面这个实例通过一个工厂创建不同对象的实例，通过这种方式可以降低代码的耦合度，代码得到了很大程度的扩展，以前要创建Apple 对象需要通过new 关键字创建Apple 对象，如果我们也要创建Orange 对象呢？是不是也要通过new 关键字创建实例并向上转型为Fruit ，这样做是麻烦的。

现在我们直接有一个工厂，你只要在配置文件中配置你要创建对象的信息，你就可以创建任何类型你想要的对象，是不是简单很多了呢？可见反射机制的价值是很惊人的。

Spring 中的 IOC 的底层实现原理就是反射机制，Spring 的容器会帮我们创建实例，该容器中使用的方法就是反射，通过解析xml文件，获取到id属性和class属性里面的内容，利用反射原理创建配置文件里类的实例对象，存入到Spring的bean容器中。

 参考书籍：   
 《Java 编程思想》