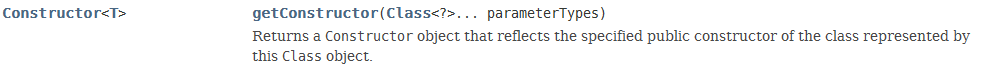
Java中的Class类和反射机制

1. **Class<T>类**：是直接继承于java.lang.Object包的，存在于java.lang包中。**Class类是**用于描述类的**字节码文件**（class文件）**Class类**是**final**类，不可被继承。任何一个类都具有一个字节码文件，且都具有一个属性class，获取该字节码文件，这个文件就是对应的这个类的**Class对象**。**每个类都只能有一个Class对象**。Class<T>类本身**没有字段也没有公共的构造方法**，在Java虚拟机加载类时，自动创建该类的Class对象，但是有很多**成员方法**。
2. **一个类的Class对象就像是一个图纸一样，可以根据这个图纸构造出该类的任何部分，包括构造方法、成员方法、成员变量、注释（Annotations）。**

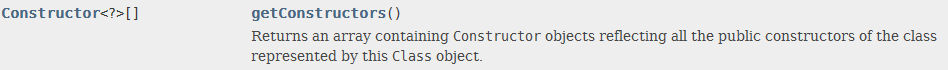
任何class对象的开头都是四个相同的字节CA FE BA BE …，便于识别出是class文件。

1. **Java中的反射机制**：指的是在运行状态中，对于**任何一个类（指的就是class文件）**，都能够知道这个类的**所有属性**（字段）和**方法**（**类名name、构造方法constuctor、字段field、成员方法method**），对于**该类的任意一个对象**都可以调用它的任意一个属性和方法；**这种动态地获取信息以及动态地调用成员方法的功能称为Java语言的反射机制。**简单的说，**动态获取类中信息就是java反射**。可以理解为**对类的解剖**。
2. 反射机制这部分学习主要有以下几大重点知识：
3. 反射机制的概念；
4. 获取**Class对象**的三种方法，尤其是**第三种**；
5. 通过Class对象获取**构造方法（Constructor）、字段（Field）、一般方法（Method）**；
6. Class<T>类提供了获取字节码文件中的**所有内容**的方法。字节码中的内容有**类名name、字段field、构造方法constuctor、成员方法method。Java中的反射机制就是依赖这个Class对象实现的。想要对一个类文件进行解剖，只要获取到该类的字节码文件对象即可。**
7. **Class<T>类的成员方法**：
8. **获取类的构造方法：**

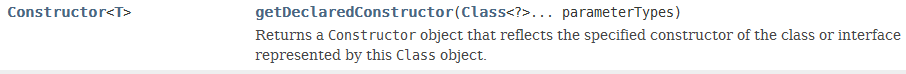
**a) getConstructor（parameterTypes）**方法：获取某个特定的构造方法，返回一个**Constructor对象**。



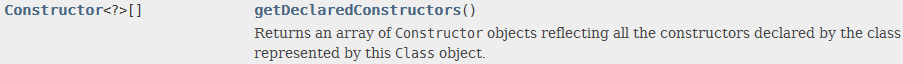
1. getConstructors()方法：返回的是**public修饰的构造方法**，利用**构造器数组**存放；



1. getDeclareConstructor（**parameterTypes**）：返回一个构造方法

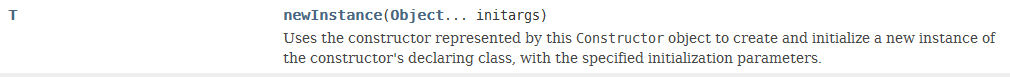


1. getDeclareConstructors()方法：返回的是**所有的构造方法**，包括private修饰的，利用**构造器数组**存放。



1. **创建该类对象的方法：**

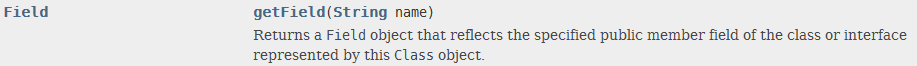
**一般是首先获取特定的构造方法对象，然后调用newInstance（**[Object](mk:@MSITStore:E:\JAVA\JAVA%20JIHE\API帮助文档\CHM帮助文档\jdk-8u60-docs-all_API.chm::/api/java/lang/Object.html)... initargs**）方法**：注意返回**T，需要进行强制转型**。



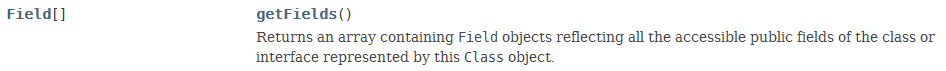
**直接利用类文件对象的newInstance（）方法，其实就是调用一个空参数构造方法（默认的构造方法）生成一个对象，此类若没有空参数的构造方法，则就会抛出InstantiationException 实例化异常。**



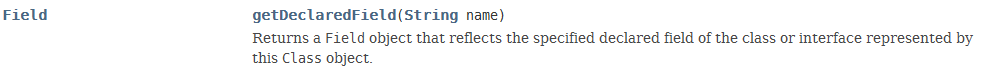
1. **获取该类的字段（Field）的方法：**
2. **getField(String name)方法： 只可以获取某一个public修饰的字段，返回一个Field对象。**



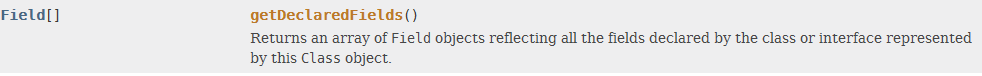
1. **getFields()方法:获取所有的public权限的字段，利用Field数组存放。**



1. **getDeclareField(String name)方法**：获取某一个字段，无论是public、protected、Default、private，都可以获取。返回一个**Field对象**。

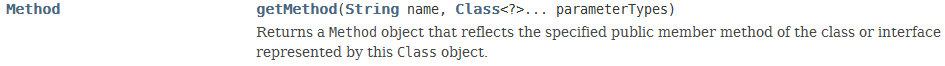


1. **getDeclareFields() 方法**：获取该类的所有字段，无论是public、protected、Default、private，存放到**Field数组**中。

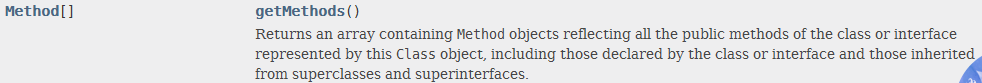


1. **获取该类成员方法（Method）的方法**：
2. **getMethod(String name，parameterTypes)**：参数是**方法名和方法参数列表**，返回一个**Method对象**。**只可以获取public权限的成员方法**。

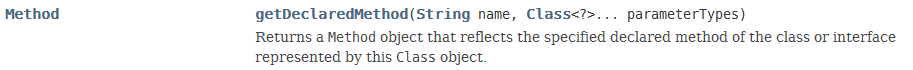
**方法无参数时，用null声明。**



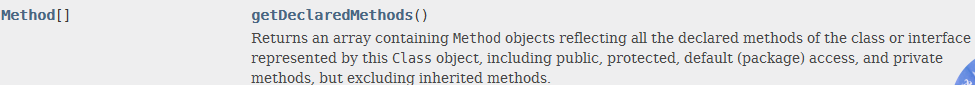
1. **getMethods()方法：****放回一个Method数组，只可以获取public权限的成员方法。**



1. **getDeclareMethods(String name，parameterTypes)方法：**参数是**方法名和方法参数列表**，返回一个**Method对象。可以获取所有权限的指定的方法。**



1. **getDeclareMethods（）方法：放回一个Method数组，可以获取所有权限的指定的方法。**

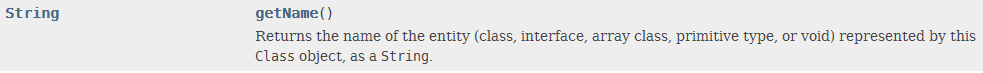


1. **获取该类名字的方法：**

**getName（）方法：放回一个字符串即类的全名 ： 包名.类名。**

**如：**System.out.println(Class.forName("java.lang.String").getName());

**//输出java.lang.String**



1. 获取**Package对象**的方法：**返回一个Package对象**。



1. **获取Class对象的三种方式：**
2. **利用Object类中的getClass方法：返回该类的Class对象；**

这种方法的前提是**必须首先创建该类的一个对象**。

**例子：Class clazz1 ="nihao".getClass();**

**Class clazz2 = "iloveyou".getClass();**

**System.out.println(clazz1 == clazz2);//返回结果是true，因为两个对象都是来自同一个类，一个类只有一个字节码文件对象。**

1. **任何数据类型都具备一个静态属性 .class 来获取其对应的Class对象；**

**这种方法**相对简单，但是**还是需要用到类中的静态成员，还是不够扩展**。

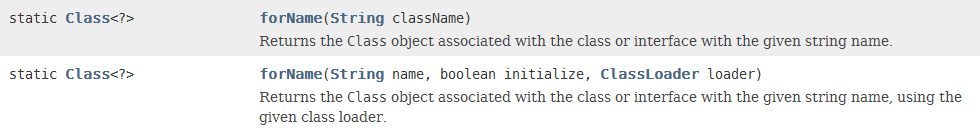
例子：Class clazz1 = String.class;

Class clazz2 = String.class;

System.out.println(clazz1 == clazz2); **//返回结果是true，因为是用的一个类**

1. **利用Class类的静态方法forName（String classname），返回指定类的字节码文件对象。**

**这种方式只要给定类名(字符串)就可以获取该类的Class对象，更为方便扩展。**



注意：**类名classname**必须是**全名**，即**包名.类名。**

例子：返回一个线程类的字节码文件对象：



1. **创建一个类的对象的方法**：
2. **利用new的方式**：创建对象前，先根据**类名**寻找该类的**字节码文件**，并加载进内存，并**创建字节码文件对象**，然后**该类的对象**。

java.lang.String str1 = new java.lang.String();

1. 利用**Class对象**的**newInstance方法**：这个方法只**局限于**类具有空参数的构造方法。

Object obj = Class.forName("java.lang.String").newInstance();

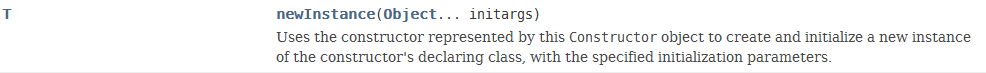
如：

PersonPP pp1 = new PersonPP();

Class clazz = Class.forName("datetest.PersonPP");//必须是类的全名

PersonPP pp2 = (PersonPP) clazz.newInstance();

1. **先获取类的构造方法，其次利用获取的构造方法Constructor对象调用Constructor类的newInstance（参数）方法创建对象。**



**注意：Class类和Constructor类都具有newInstance方法，只不过Class类的newInstance方法是空参数的，而Constructor类的newInstance方法需要添加参数，调用的构造方法不同。**

**如：PersonPP pp4 = new PersonPP(23,"zhaohong");**

**Constructor con = clazz.getConstructor(int.class,String.class);**

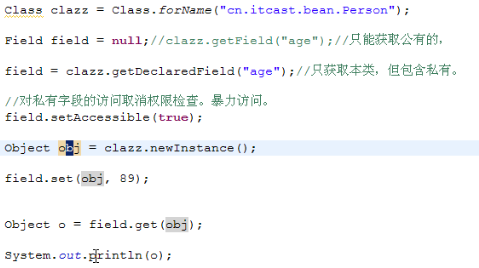
**PersonPP pp2 = (PersonPP) con.newInstance(23,"zhaohong");**

1. **反射包**：**java.lang.reflect包**。这个包存放了**反射机制**所采用的类。这个包中有个类叫做**AccessibleObject**可访问对象；**AccessibleObject有两个直接子类，Field和**Executable类，Executable类下面又两个直接子类**Constructor 和Method**类。**AccessibleObject**是**Field**类的**直接父类**，是**Constructor 、Method**类的**间接父类**，Constructor和Method直接继承于**Executable 类**，而Executable类直接继承于**AccssibleObject类**。

**AccssibleObject类有个方法：setAccessible（boolean flag），如果参数为true，则会使调用此方法的Field对象、Method对象、Constructor对象等不考虑权限问题，这种方式称为暴力访问。不建议采用暴力访问。**



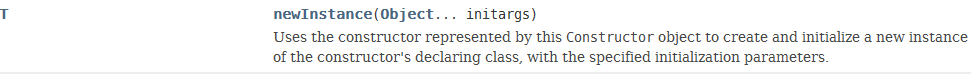
所有带有**Declare**的方法都是相当于**暴力访问**或**强制访问**。如：



1. **Constructor类**：

注意：构造方法是依赖于**类**的，与具体对象无关，所以使用Constructor类的方法**无需指明类的那个对象**，与**Field、Method**相区别。

1. **newInstance（）方法**：返回一个对象。



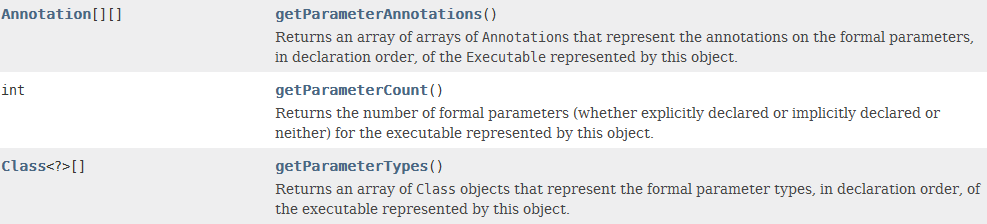
1. **getName（）方法**：返回一个字符串。



1. 获取修饰符:getModifiers()**方法**：返回int值。



1. 获取参数的**一些属性方法**：



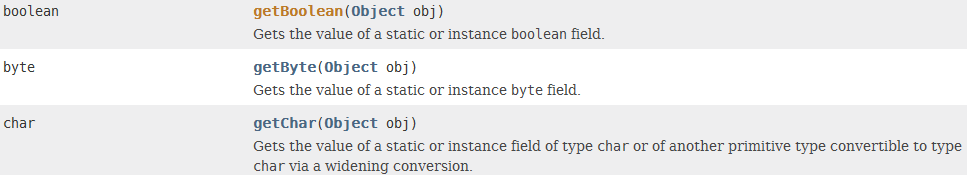
1. **Field类**：

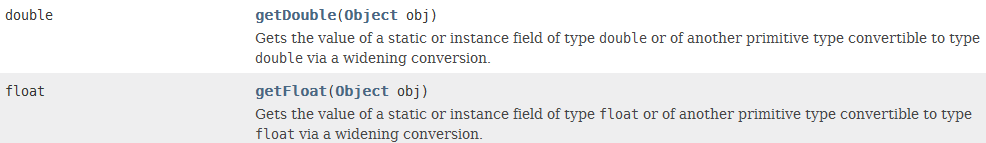
注意：Field是类的字段，显然是该类的每个对象都具有不同的字段值，故在获取字段值或设置字段值时，需要指定具体的那个对象。

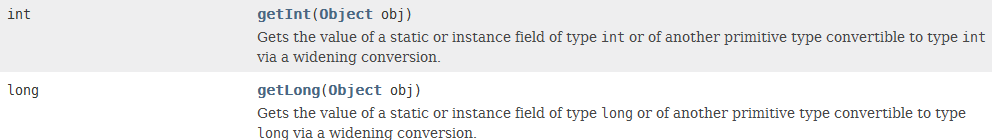
1. 最重要的方法为两类：**get方法和set方法**。由于有字段的类型千变万化，所以利用**Object类**指明，除此之外，由于8种基本数据类型不是对象，所以单独列出，但是引入了对象包装类后，有了自动装箱和拆箱功能后，利用**Object类**都可以实现。
2. **get（Object obj）方法**：参数就是该类的一个对象，返回值为字段值，利用Object接收。

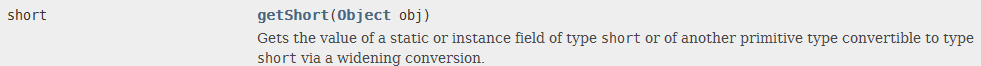


**基本数据类型的getXxxx（Xxx xxx）方法**：

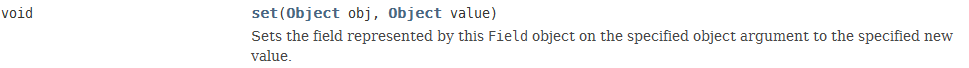








1. **set(Object obj,Object value)方法**：第一个参数为访问的对象，第二个参数为**要设置的该字段的值**。返回值为空。



基本数据类型的**setXxxx（Object obj，Xxx xxx）**方法：



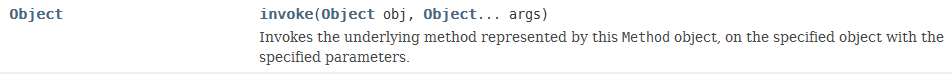
1. **getName（）方法**：由于字段名字不依赖于具体对象，所以不需要指定某个对象，**返回字符串**。



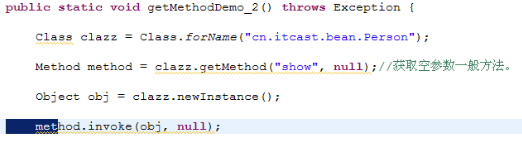
1. **Method类**：

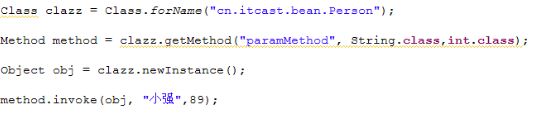
**注意：成员方法也是依赖于具体的对象的，所以执行方法时有必要指定是该类的哪个对象。**

1. **invoke（Object obj，Object…args）方法：执行当前方法,方法的返回值利用Object接收。**



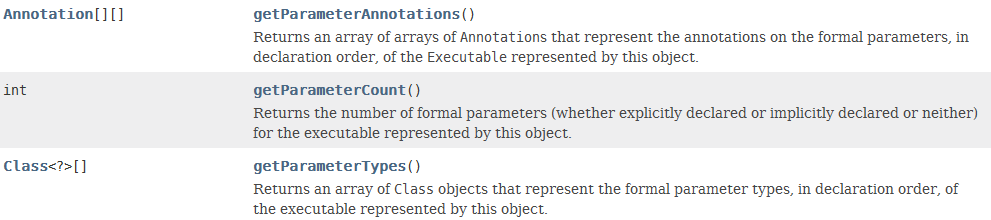
如：





1. 很多方法和Constructor类似，因为都是方法，有一些共性，如方法名、参数个数、参数类型等。





1. 获取**返回类型**：



1. **annotation 英 [ænə'teɪʃ(ə)n] 美 [,æno'teʃən] n. 注释；注解；释文**

就是如**@Override**等。

注释如**@Override用在方法前面，用于限定这个方法必须是重写的，若方法不是重写的，则此处就会报出错误。用途：可以降低错误率。**

1. **练习题**：
2. **反射应用：**

利用**配置文件**存放**键值对**，（值就是**类的名字**），通过不断添加并获取配置文件中的信息，就可以不断访问到**添加进来的类及其方法**。

下面代码就是**主板访问接口设备，如声卡、网卡**等。

**添加了新的设备，只需要修改配置文件（manifest.properties）即可，无需更改代码，就可以运行相应的设备了。、**

**Test类：**

public class MultiPicTest {

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

File file = new File("manifest.properties");

Properties prop = new Properties();

InputStream in = new FileInputStream(file);

prop.load(in);

MainBoard mb = new MainBoard();

String str1 = "Camera";

String str2 = "Viedo";

String str3 = "USB";

usePIC(prop,str1,mb);

usePIC(prop,str2,mb);

usePIC(prop,str3,mb);

}

private static void usePIC(Properties prop, String str,MainBoard mb) throws Exception {

String str1 = (String) prop.get(str);

Class<?> clazz = Class.forName(str1);

PIC pic = (PIC)clazz.newInstance();

mb.usePIC(pic);

}

Mainboard主板：

**public** **class** MainBoard {

**public** MainBoard() {

**super**();

System.***out***.println("The MainBoard is running.");

}

**public** **void** run() {

System.***out***.println("The MainBoard is running.");

}

**public** **void** openPIC(PIC pic) {

pic.open();

}

**public** **void** closePIC(PIC pic) {

pic.close();

}

**public** **void** runPIC(PIC pic) {

pic.run();

}

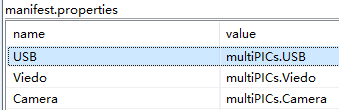
**public** **void** usePIC(PIC pic) {

pic.use();

}

}

manifest.prpperties:



PIC 接口:

**public** **interface** PIC {

**public** **void** use();

**public** **void** open();

**public** **void** close();

**public** **void** run();

}

实现的硬件设备：

**public** **class** USB **implements** PIC {

@Override

**public** **void** open() {

System.***out***.println("USB is open.");

}

@Override

**public** **void** close() {

System.***out***.println("USB is closed.");

}

@Override

**public** **void** run() {

System.***out***.println("USB is running.");

}

@Override

**public** **void** use() {

**this**.open();**this**.run();**this**.close();

}

}

**Viedo、Camera同USB**。