# 反射的概念及在[Java](http://lib.csdn.net/base/javaee" \o "Java EE知识库" \t "_blank)中的类反射

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。在计算机科学领域，反射是一类应用，它们能够自描述和自控制。这类应用通过某种机制来实现对自己行为的描述和检测，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。

在Java中的反射机制，被称为Reflection（大家看到这个单词，第一个想法应该就是去开发文档中搜一下了）。它允许运行中的Java程序对自身进行检查，并能直接操作程序的内部属性或方法。

Reflection机制允许程序在正在执行的过程中，利用Reflection APIs取得任何已知名称的类的内部信息，

包括：

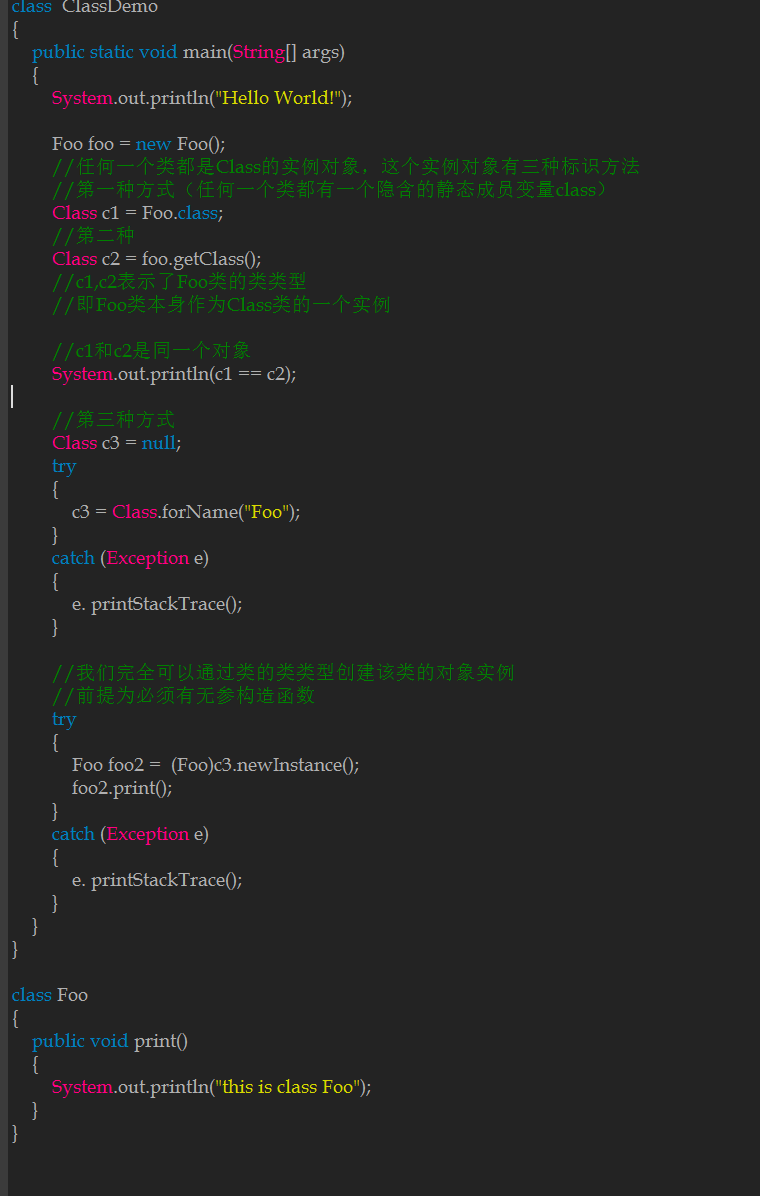
package、 type parameters、 superclass、 implemented interfaces、 inner classes、 outer classes、 fields、 constructors、 methods、 modifiers等，并可以在执行的过程中，

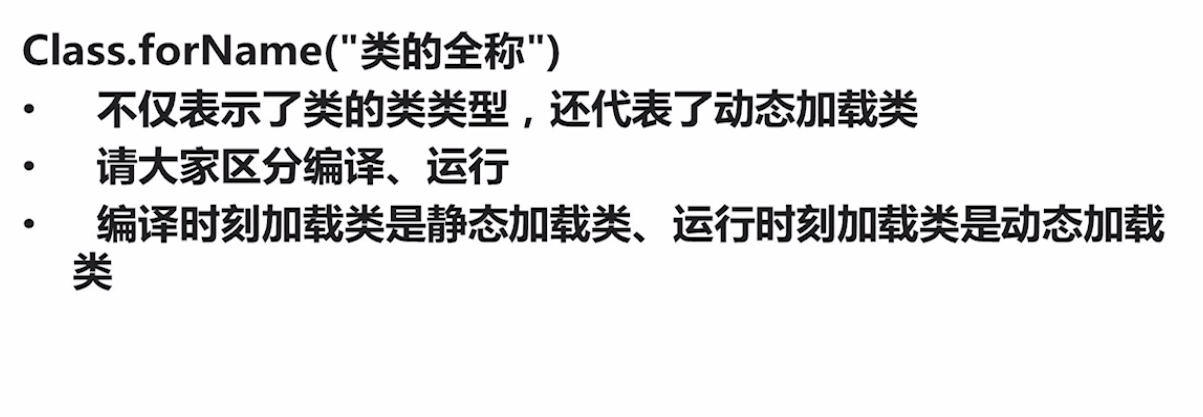
1. 动态生成Instances、
2. 变更fields（成员变量）内容
3. 唤起methods。

# Class类的使用

java里万事万物皆为对象，类也是对象，是Class类，这个对象我们称为该类的类类型。但静态成员（静态成员属于类不属于对象）、普通的数据类型不是对象

类是java.lang.Class类的实例对象





## 静态加载类（编译时刻加载类）：

new创建对象是静态加载类，在编译时刻就需要加载可能用到的类。即：编译时刻类中有错误就会导致程序编译失败无法运行，而出错的地方可能在程序实际运行时并不会用到，通过动态加载类可以解决该问题

## 动态加载类（运行时刻加载类）：

class CLassLoadTrends

{

public static void main(String[] args)

{

try

{

Class c = Class.forName(args[0]);

Officeable officeable = (Officeable)c.newInstance();

officeable.start();

}

catch (ClassNotFoundException e)

{

e.printStackTrace();

}

catch (InstantiationException e)

{

e.printStackTrace();

}

catch (IllegalAccessException e)

{

e.printStackTrace();

}

}

}

class Word implements Officeable

{

public void start()

{

System.out.println("Word is start");

}

}

class Excel implements Officeable

{

public void start()

{

System.out.println("Excel is start");

}

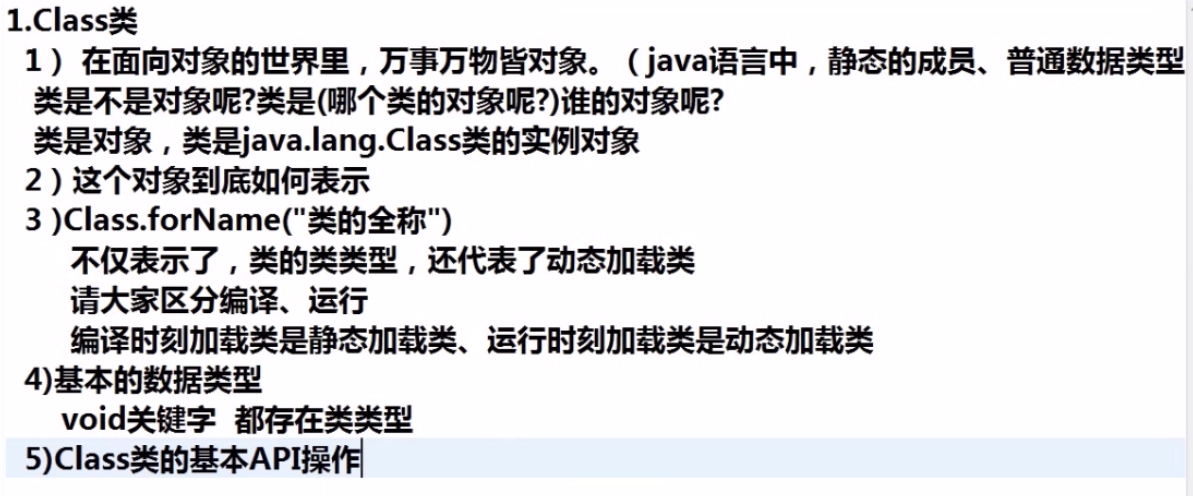
}

interface Officeable

{

public void start();

}



public static void main(String[] args)

{

//int的类类型

Class c = int.class;

//String类的类类型（String类的字节码）

Class c1 = String.class;

//基本数据类型与其对应的类的类类型不相等

Class c2 = void.class;

Class c3 = Void.class;

System.out.println(c1.getName());

System.out.println(c2.getName());

System.out.println(c3.getSimpleName());

}

# 方法的反射

## Java.lang.reflect.Method类封装了对类的方法的操作

import java.lang.reflect.Method;

class ClassDemo2

{

public static void main(String[] args)

{

new ClassUtil().printClassMessage("sdfa");

}

}

class ClassUtil

{

/\*\*

\*打印类的信息，类的成员函数，成员变量

\*/

public void printClassMessage(Object obj)

{

//首先获取类类类型，传递的是哪个子类的对象，c就是该子类的类类型

Class c = obj.getClass();

//获取类的名称

String className = c.getName();

System.out.println("Class "+className);

//获得类的方法，一个成员方法就是一个Method对象

//getMethod方法获得所有的public方法包括继承于父类的

//getDeclaredMethods方法获得所有该类自己的方法，与权限无关

Method[] ms = c.getMethods();

Method[] mds = c.getDeclaredMethods();

for (int i=0;i < ms.length ;i++ )

{

//得到方法的返回值类型的类类型

Class returnType = ms[i].getReturnType();

//得到方法的名称

String name = ms[i].getName();

System.out.print(name+" (");

//获得参数类型\_\_的到参数列表的类型的类类型

Class[] paramTypes = ms[i].getParameterTypes();

for (Class class1 : paramTypes)

{

System.out.print(class1.getName()+" ");

}

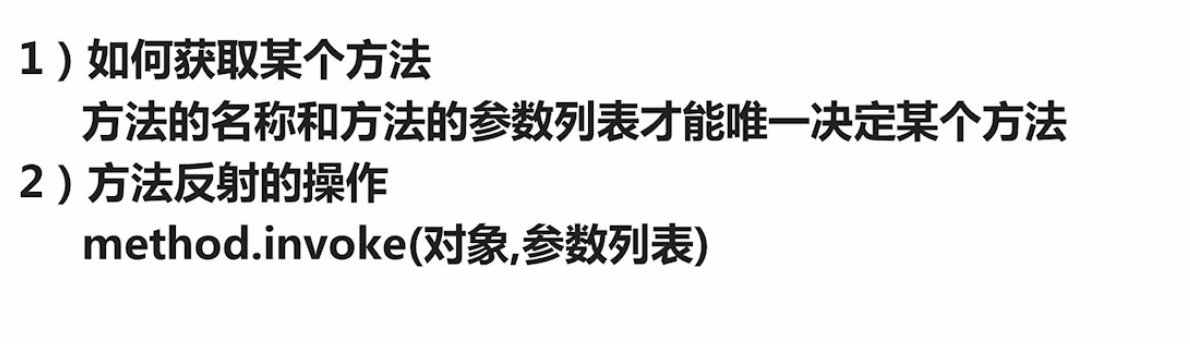
System.out.print(")");

System.out.println();

}

}

}



## 获得方法对象并通过反射调用方法

import java.lang.reflect.\*;

import java.util.\*;

class ClassDemo2

{

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

{

//System.out.println(new Test().getNum(3,"asdf"));

Test test = new Test();

Class clt = test.getClass();

//获取方法 名称和参数列表决定（方法可能重载）

//getMethod()

//getDeclaredMethod()

Method m = clt.getMethod("getNum",int.class,String.class);

Method m2 = clt.getMethod("getNum");

//方法的反射操作

//用Method的实例调用方法与一般的方法调用效果完全相同

int intRe = (int)m.invoke(test,789,"sadf");

System.out.println(intRe+"");

System.out.println((int)m2.invoke(test));

}

}

class Test

{

public int getNum(int a,String str)

{

str = str+a;

return str.length();

}

public int getNum()

{

return -1;

}

}

## 通过反射了解泛型的本质：

反射的操作都是编译之后的操作，编译之后集合的泛型是去泛型化的

Java中的泛型是为了防止错误输入（赋值），只在编译阶段有效，绕过编译就无效了。

验证：通过方法的反射绕过编译

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

{

List list = new ArrayList<String>();

list.add("hello");

Class li = list.getClass();

Method m = li.getMethod("add",Object.class);

//绕过了编译就绕过了泛型限制

Integer integer = 100;

m.invoke(list,integer);

System.out.println(list);

}

# 成员变量的反射

## Java.lang.reflect.Field类封装了对java类的成员变量的操作

import java.lang.reflect.Field;

public void printClassMessage(Object obj)

{

//首先获取类类类型，传递的是哪个子类的对象，c就是该子类的类类型

Class c = obj.getClass();

//获取类的名称

String className = c.getName();

System.out.println("Class "+className);

Field[] f = c.getDeclaredFields();

for (Field field:f )

{

System.out.println(field.getType().getName()+" "+field.getName());

}

}

# 构造函数的反射

## Java.lang.reflect.Constructor类封装了对java类的构造函数的操作

public void printClassMessage(Object obj)

{

//首先获取类类类型，传递的是哪个子类的对象，c就是该子类的类类型

Class c = obj.getClass();

//获的自己声明的构造函数

Constructor[] con = c.getDeclaredConstructors();

for (Constructor cons : con)

{

Class[] paramTypes = cons.getParameterTypes();

System.out.print(cons.getName()+"(");

for (Class cl : paramTypes)

{

System.out.print(cl.getName()+",");

}

System.out.println(")");

}

}

# Java类加载机制