**Java反射机制**

1、JAVA类的结构：

Java类的结构和对应的反射类

包声明 ------------- Package

类声明 ------------- Class

构造函数声明 ------------- Constructor

字段声明 ------------- Field

方法声明 ------------- Method

静态代码块和代码块

2、什么是Java反射机制？

①Java反射机制就是把Java类按结构分解,每一部分对应特定的反射类

②Java反射机制允许运行时加载、探知和使用在编译期间完全未知的classes。换句话说，Java程序可以加载一个运行时才得知名称的class，获悉其完整结构信息

3、

//一个类的结果通常包括: 字段, 构造函数 ,方法 Class

//一个.class文件,就对应一个Class类的对象

**public** **class** Person {

// 字段 , 属性 , 全局变量：对于Java反射来说,字段对应一个反射类 : Field

**private** String name;

**private** **int** age;

// 无参数构造函数：对于Java反射来说,构造函数对应一个反射类 : Constructor

**public** Person() {

}

// 有参数构造函数

**public** Person(String name, **int** age) {

**this**.name = name;

**this**.age = age;

}

// 普通方法：对于Java反射来说,普通方法对应一个反射类 : Method

**public** **void** sayHello() {

System.*out*.println("你好 ,我叫 " + name + " , 今年 " + age + " 岁了");

}

// 声明了抛出异常的方法

**public** **void** sing(String songName) **throws** Exception {

**if** ("两只老虎".equals(songName)) {

System.*out*.println("sing ~ 两只老虎两只老虎跑得快跑得快...");

} **else** {

**throw** **new** Exception("对不起, 我不会唱 " + songName);

}

}

}

**Java反射机制之Class类**

1、主要方法：

Class forName(String className) //通过类的全类名加载该类

Constructor getConstructor(Class... parameterTypes)

Constructor[ ] getConstructors() //获得所有的public构造函数,包括父类的

Constructor getDeclaredConstructor(Class... parameterTypes)

Constructor[ ] getDeclaredConstructors() //获得自己声明的所有构造函数

Field getField(String name)

Field[ ] getFields() //获得所有的public字段,包括父类的

Field getDeclaredField(String name)

Field[ ] getDeclaredFields() //获得自己声明的所有字段,包括private的

Method getMethod(String name, Class... parameterTypes)

Method[ ] getMethods() //获得所有的public方法

Method getDeclaredMethod(String name, Class... parameterTypes)

Method[ ] getDeclaredMethods() //获得自己声明的所有方法

2、其他方法：

Package getPackage() //获得此类的包的对象

int getModifiers() //获得此类的修饰符

String getName() //获得此类的名称

Object newInstance() //使用无参数构造方法创建此类的对象

boolean isArray() //判断此类是否是数组类型

boolean isInterface() //判断此类是否是接口

boolean isEnum() //判断此类是否是枚举类型

3、//演示JAVA反射类之Class（Class与class不同，前者是类，后者是关键字）

**public** **class** ClassTest1 {

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

// 获得一个类的class对象

// 方式1

Class class1 = Person.**class**;

// 方式2

Person person1 = **new** Person();

Class class2 = person1.getClass();

// 方式3

Class class3 = Class.*forName*("com.xbf.reflect.Person");

// getName()：获取类的全名，Class类不是静态类，所以它的方法要有实例来调用

System.*out*.println(class1.getName());

System.*out*.println(class2.getName());

System.*out*.println(class3.getName());

// 因为一个类只加载一次，所以一个类对象只会创建一次，其他的都是指向

System.*out*.println(class1 == class2);// 输出为true，说明这些变量指向同一个Class对象

// getPackage()，获得并返回一个包对象

System.*out*.println(class1.getPackage());

// getSimpleName()，获取类的简单名称

System.*out*.println(class1.getSimpleName());

System.*out*

.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*getConstructors()\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// getConstructors()，获取类的所有public构造函数并返回由他们组成的一个数组

Constructor[] constructors = class1.getConstructors();

**for** (Constructor constructor : constructors) {// 增强for循环

System.*out*.println(constructor);

}

System.*out*

.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*getDeclaredFields()\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// getDeclaredFields()，获取获取类的所有private成员（字段）并返回由他们组成的一个数组

Field[] fields = class1.getDeclaredFields();

**for** (Field field : fields) {

System.*out*.println(field);

}

System.*out*

.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*getMethods()\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

// getMethods()，获取类的所有public方法（包括父类的）并返回由他们组成的一个数组

Method[] methods = class1.getMethods();

**for** (Method method : methods) {

System.*out*.println(method);

}

System.*out*

.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

}

}

**Java反射机制之Constructor类**

1、方法：

Object newInstance(Object... initargs) //创建一个对应类的对象

Class[ ] getParameterTypes() //获得所有参数的类型

String getName() //获得构造函数名称

int getModifiers() //获得此构造函数的修饰符

Class[ ] getExceptionTypes() //获得此构造函数声明抛出的异常类型

Object getAnnotation(Class annotationClass) //获得注释对象

2、//演示JAVA反射之Constructor

**public** **class** ConstructorTest1 {

**public** **static** **void** main(String[] args) **throws** ClassNotFoundException,

NoSuchMethodException, SecurityException, InstantiationException,

IllegalAccessException, IllegalArgumentException,

InvocationTargetException {

Class class1 = Class.*forName*("com.xbf.reflect.Person");// 创建Class对象

Constructor constructor = class1.getDeclaredConstructor();// 获取private无参构造函数

// newInstance()，通过构造函数获得Object对象

Object object = constructor.newInstance();

Person person = (Person) object;// 强制类型转化

person.sayHello();

// 传入构造函数的参数类型获取对应的有参的构造函数

Constructor constructor2 = class1.getDeclaredConstructor(String.**class**,

**int**.**class**);

Object object2 = constructor2.newInstance("谢宝发", 20);// 使用有参构造函数传入参数得到一个Object对象

Person person2 = (Person) object2;

// 不能这样写：Class

// class=(Person)object2;，原因是object2是被Person实例化的Objec变量，最多被转化为子类Person

// ，而不能再被转化为其他子类，Object是一切类（包括Class）的父类

person2.sayHello();

}

}

**Java反射机制之Field类**

1、方法：

Class getType() //获得此字段的类型

Class getDeclaringClass() //获得声明此字段的类的类型

String getName() //获得此字段的名称

int getModifiers() //获得此字段的修饰符

Object get(Object obj) //获得此字段在指定对象中的值

boolean getBoolean(Object obj) //获得此字段在指定对象中的布尔值

byte getByte(Object obj) //获得byte值

void set(Object obj, Object value) //设置此字段在指定对象中的值

void setBoolean(Object obj, boolean z)

2、

//演示java反射类之Field

**public** **class** FieldTest1 {

**public** **static** **void** main(String[] args) **throws** InstantiationException,

IllegalAccessException, NoSuchFieldException, SecurityException {

Class class1 = Person.**class**;

Field[] fields = class1.getDeclaredFields();// 获得所有声明的字段

**for** (Field field : fields) {

System.*out*.println(field);

}

Person person = (Person) class1.newInstance();

Field nameField = class1.getDeclaredField("name");// getDeclaredField(String

// name)；获得指定的声明的字段

Field ageField = class1.getDeclaredField("age");

// setAccessible()，通过字段取得访问和修改对象的私有对象的权限

nameField.setAccessible(**true**);// true表示获得

ageField.setAccessible(**true**);

// set()，通过字段设置指定对象中该字段的值

nameField.set(person, "周杰伦");

ageField.set(person, 36);

// get()，通过字段获得指定对象中该字段的值

Object object = nameField.get(person);

System.*out*.println(object);

person.sayHello();

}

}

**Java反射机制之Method类**

1、方法：

Class getDeclaringClass() //获得声明此方法的类的Class对象

Class[ ] getExceptionTypes() //获得此方法声明抛出的异常类型

int getModifiers()//获得此方法的修饰符

String getName() //获得此方法的名称

Class[ ] getParameterTypes() //获得此方法的参数的类型

Class getReturnType() //获得此方法的返回值类型

Object invoke(Object obj, Object... args) //使用指定对象调用此方法

2、

//测试JAVA反射之Method

**public** **class** MethodTest1 {

**public** **static** **void** main(String[] args) **throws** NoSuchMethodException,

SecurityException, InstantiationException, IllegalAccessException,

IllegalArgumentException, InvocationTargetException {

// 涉及到反射肯定要创建Class对象

Class class1 = Person.**class**;

Method[] methods = class1.getDeclaredMethods();// 获得所有声明的方法

**for** (Method method : methods) {

System.*out*.println(method);

}

Method sayHello = class1.getDeclaredMethod("sayHello");

// Person person = (Person) class1.newInstance();

Person person = **new** Person("乔布斯", 55);

// invoke()，通过方法调用指定类对象的该方法

sayHello.invoke(person);// 效果一样

person.sayHello();// 效果一样

// getReturnType()，获得方法的返回值类型并返回一个Class对象

Class returnType = sayHello.getReturnType();

System.*out*.println(returnType);

// getModifiers()，通过方法获取其修饰符并返回int

**int** mod = sayHello.getModifiers();

System.*out*.println(Modifier.*isPublic*(mod));

// 调用有参方法

Method singMethod = class1.getDeclaredMethod("sing", String.**class**);

singMethod.invoke(person, "两只老虎");

}

}

**Java反射机制之Modifier修饰符**

static int ABSTRACT

static int FINAL //静态常量

static int INTERFACE

static int NATIVE

static int PRIVATE

static int PROTECTED

static int PUBLIC

static boolean isAbstract(int mod)

static boolean isFinal(int mod) //判断方法

static boolean isInterface(int mod)

static boolean isNative(int mod)

static boolean isPrivate(int mod)

static boolean isProtected(int mod)

static boolean isPublic(int mod)

**Java反射机制之AccessibleObject**

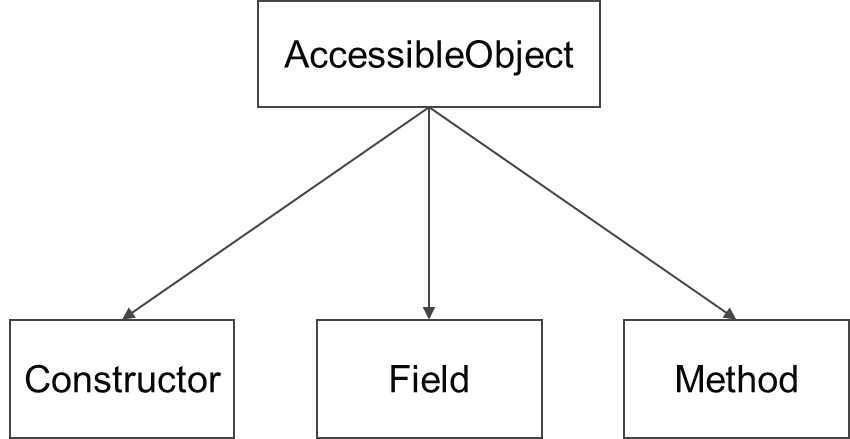
1、

取消反射对象Constructor, Field , Method 的访问限制

void setAccessible(boolean flag)

java 对象的序列化会使用到此特性

2、

****

**Java反射机制之结束语**

1、Java 反射可以做什么？

①Java对象的序列化

②封装框架提供通用功能

③为程序提供扩展功能

2、案例 : 餐厅菜单可配置化

**public** **class** Restaurant {

// 定义一个list对象存储菜谱

**private** List<Food> foods = **new** ArrayList<Food>();

// Restaurant的构造函数

**public** Restaurant() **throws** IOException {

init();// 将init方法单独封装是为了更好地被阅读

}

**public** **void** init() {// 初始化方法

Properties properties = **new** Properties();

FileInputStream fileInputStream;

**try** {

fileInputStream = **new** FileInputStream("food.properties");

properties.load(fileInputStream);

// 获取value：类的路径

Collection<Object> collections = properties.values();// properties是一个Map类，可以使用Map的方法

**for** (Object foodType : collections) {

Class class1 = Class.*forName*((String) foodType);// 得到Class对象

Food food = (Food) class1.newInstance();// 创建food对象

foods.add(food);// 将food对象依次添加到foods这个list中

}

} **catch** (Exception e) {

**throw** **new** RuntimeException("菜谱加载异常" + e);

}

}

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

Restaurant restaurant = **new** Restaurant();

System.*out*.println("欢迎光临~~");

System.*out*

.println("\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*");

restaurant.show();

}

**public** **void** show() {// 展示菜谱

**for** (Food food : foods) {

System.*out*.println(food.getFoodName());

}

}

}