java 反射

> 之前学的java反射是从p师傅的java安全里面学习的，只能说对自己印象不是特别深刻，而且最近自己想实现一个代码混淆功能，并且通过反射去执行没有写出来，然后就准备在学习一次，温故知新。

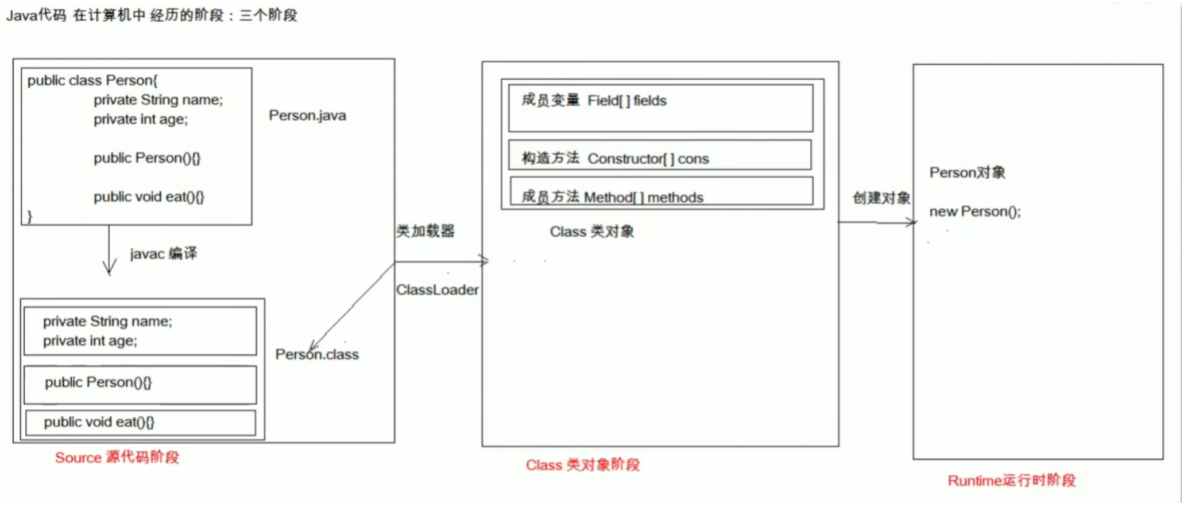
反射

反射就是将各个组成部分封装为其他对象，这就是反射机制。

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和动态调用对象的方法的功能称为java语言反射机制。

优点：可以在程序运行过程中，操作这些对象（比如说idea的代码提示功能），动态加载类，可以解耦，提高代码灵活性。（反射是框架的灵魂）
缺点：反射相当于一系列解释操作，通知JVM要做的事，性能比直接的java代码要慢的多，并且动态操作改变类的属性同时增加安全隐患。

java代码在计算机中经历阶段



如何使用反射?

要使用反射必须首先要获得Class对象，那么如何获得Class对象?

获得Class对象的方式。

```
1.Class.forName("全类名");  //将字节码文件加载内存，返回Class对象
//多用于配置文件，直接写全类名
2.类名.class;            //通过类名的属性class获得
//多用于参数传递
3.对象.getClass();          //getClass()方法在Object类中定义
//多用于对象获得字节码的方式
```

代码演示

Person

```java
package reflect.user;

public class Person {
    private String name;
    public int age;

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }

    public Person(){
    }
}
```

```java
package reflect;

import reflect.user.Person;

public class demo1 {
    /** 获得Class对象的方式。
     * 1.Class.forName("全类名");  //将字节码文件加载内存，返回Class对象
     * 2.类名.class;            //通过类名的属性class获得
     * 3.对象.getClass();        //getClass()方法在Object类中定义
     */
    public static void main(String[] args) throws Exception {
        //1.Class.forName("全类名");
       Class clazz = Class.forName("reflect.user.Person");
        System.out.println(clazz);
        //2.类名.class;
        Class clazz2 = Person.class;
```

```
        System.out.println(clazz2);
        //3.对象.getClass();
        Person person = new Person();
        Class clazz3 = person.getClass();
        System.out.println(clazz3);

        // == 比较3个对象
        System.out.println(clazz==clazz2);
        System.out.println(clazz == clazz3);
        //同一个字节码文件（*.class）在一次程序运行过程中只会加载一次
        //不管通过什么方法获得Class对象都是同一个
    }
}
```

现在我们获得到了Class对象，就需要知道Class对象的功能。

```
1.获取成员变量
    * Field[] getFields()  //获得pubilc 形式的变量
    * Field getField(String name)

    * Field[] getDeclaredFields()  //获得所有的变量不考虑修饰符
    * Field getDeclaredField(String name)

2.获取构造方法
    * Constructor<?>[] getConstructors()
    * Constructor<T> getConstructor(类<?>... parameterTypes)

    * Constructor<?>[] getDeclaredConstructors()
    * Constructor<T> getDeclaredConstructor(类<?>... parameterTypes)

3.获取成员方法
    * 方法[] getMethods()
    * 方法 getMethod(String name, 类<?>... parameterTypes)

    * 方法[] getDeclaredMethods()
    * 方法 getDeclaredMethod(String name, 类<?>... parameterTypes)
4.获取类名
    * String getName()
```

ctrl+alt+v 快速付值

对成员变量的操作

我们获得了成员变量之后能做什么？

```
* Field：成员变量
    * 操作：
        1. 设置值
            * void set(Object obj, Object value)
        2. 获取值
            * get(Object obj)
        3. 忽略访问权限修饰符的安全检查
            * setAccessible(true):暴力反射
```

```
package reflect;
```

```
import reflect.user.Person;

import java.lang.reflect.Field;

public class demo2 {
    public static void main(String[] args) throws Exception {
        //1.获得Class
        Class clazz = Person.class;
        //2.获得成员变量
        Field[] fields = clazz.getFields();
        for (Field field:fields){
            System.out.println(field);
        }

        Field age = clazz.getField("age");
        Person person = new Person();
        System.out.println(age.get(person));//获得成员变量age的值

        age.set(person,10);//设置成员变量的值
        System.out.println(person);

        System.out.println("================================");

        Field[] declaredFields = clazz.getDeclaredFields();//获得所有的变量不考虑修
饰符
        for(Field declaredField:declaredFields){
            System.out.println(declaredField);
        }

        Field name = clazz.getDeclaredField("name");
        name.setAccessible(true);//忽略修饰符的安全访问，暴力反射
        name.set(person,"lx");//这样直接会报错
        System.out.println(person);
    }
}
```

对构造方法的操作

之后就是获得构造方法

```
* Constructor:构造方法
    * 创建对象：
        * T newInstance(Object... initargs)

        * 如果使用空参数构造方法创建对象，操作可以简化：Class对象的newInstance方法
```

```
package reflect;

import reflect.user.Person;

import java.lang.reflect.Constructor;

public class demo3 {
    public static void main(String[] args) throws Exception{
        //1.获得Class
        Class clazz = Person.class;
```

```
        //2.获得构造方法 创建对象
        Constructor constructor1 = clazz.getConstructor();//无参构造
        Object o1 = constructor1.newInstance();
        System.out.println(o1);

        Constructor constructor =
clazz.getConstructor(String.class,int.class);//有参构造
        Object o = constructor.newInstance("lx",10);
        System.out.println(o);
        //直接使用Class的newInstance方法
        Object o2 = clazz.newInstance();
        System.out.println(o2);
    }
}
```

对成员方法的操作

在之后就是获得成员方法，并且操作

```
* Method：方法对象
    * 执行方法：
        * Object invoke(Object obj, Object... args)

    * 获取方法名称：
        * String getName:获取方法名
```

对Person类进行一个修改

```
package reflect.user;

public class Person {
    private String name;
    public int age;

    public Person(){
    }

    public Person(String name, int age) {
        this.name = name;
        this.age = age;
    }

    @Override
    public String toString() {
        return "Person{" +
                "name='" + name + '\'' +
                ", age=" + age +
                '}';
    }

    public String getName() {
        return name;
    }

    public void setName(String name) {
        this.name = name;
```

```
    }

    public int getAge() {
        return age;
    }

    public void setAge(int age) {
        this.age = age;
    }
    public void eat(){
        System.out.println("正在吃饭");
    }
    public void eat(String food){
        System.out.println("正在吃饭"+food);
    }
    private void sleep(){
        System.out.println("睡觉是秘密的");
    }

}
```

```
package reflect;

import reflect.user.Person;

import java.lang.reflect.Method;

public class demo4 {
    public static void main(String[] args) throws Exception {
        //1.获得Class
        Class clazz = Person.class;

        //2.获得成员方法，需要 方法名，返回值，参数
        //确定一个方法需要 方法名，参数
        Method eat = clazz.getMethod("eat");
        //就需要执行方法
        Person person = new Person();
        eat.invoke(person);

        Method eat1 = clazz.getMethod("eat", String.class);
        eat1.invoke(person,".....好东西");


        Method sleep = clazz.getDeclaredMethod("sleep");//不考虑修饰符
        sleep.setAccessible(true);//暴力反射
        System.out.println(sleep.getName());
        sleep.invoke(person);

        //获得全部的方法
        System.out.println("======================");
        Method[] methods = clazz.getMethods();
        for(Method method:methods){
            System.out.println(method);
        }
    }
}
```

案例

```
* 案例：
    * 需求：写一个"框架"，不能改变该类的任何代码的前提下，可以帮我们创建任意类的对象，并且执行其中任意方法
        * 实现：
            1. 配置文件
            2. 反射
        * 步骤：
            1. 将需要创建的对象的全类名和需要执行的方法定义在配置文件中
            2. 在程序中加载读取配置文件
            3. 使用反射技术来加载类文件进内存
            4. 创建对象
            5. 执行方法
```

pro.properties文件

```
className=reflect.user.Person
methodName=eat
```

```
package reflect;

import java.io.InputStream;
import java.lang.reflect.Method;
import java.util.Properties;

public class ReflectTest {
    public static void main(String[] args) throws Exception {
        /**
         *  可以创建任意类的对象，可以执行任意方法
         *  前提：不能改变该类的任何代码。可以创建任意类的对象，可以执行任意方法
         */
        //1.加载配置文件
        //1.1创建Properties对象
        Properties pro = new Properties();
        //1.2加载配置文件，转换为一个集合
        //1.2.1获取class目录下的配置文件
        ClassLoader classLoader = ReflectTest.class.getClassLoader();//获得当前的加载器
        InputStream is = classLoader.getResourceAsStream("pro.properties");
        pro.load(is);

        //2.获取配置文件中定义的数据
        String className = pro.getProperty("className");
        String methodName = pro.getProperty("methodName");

        //3.加载该类进内存
        Class clazz = Class.forName(className);
        //4.创建对象
        Object o = clazz.newInstance();
        //5.获取方法对象
        Method method = clazz.getMethod(methodName);
        //6.执行方法
```

```
            method.invoke(o);
        }
    }
```