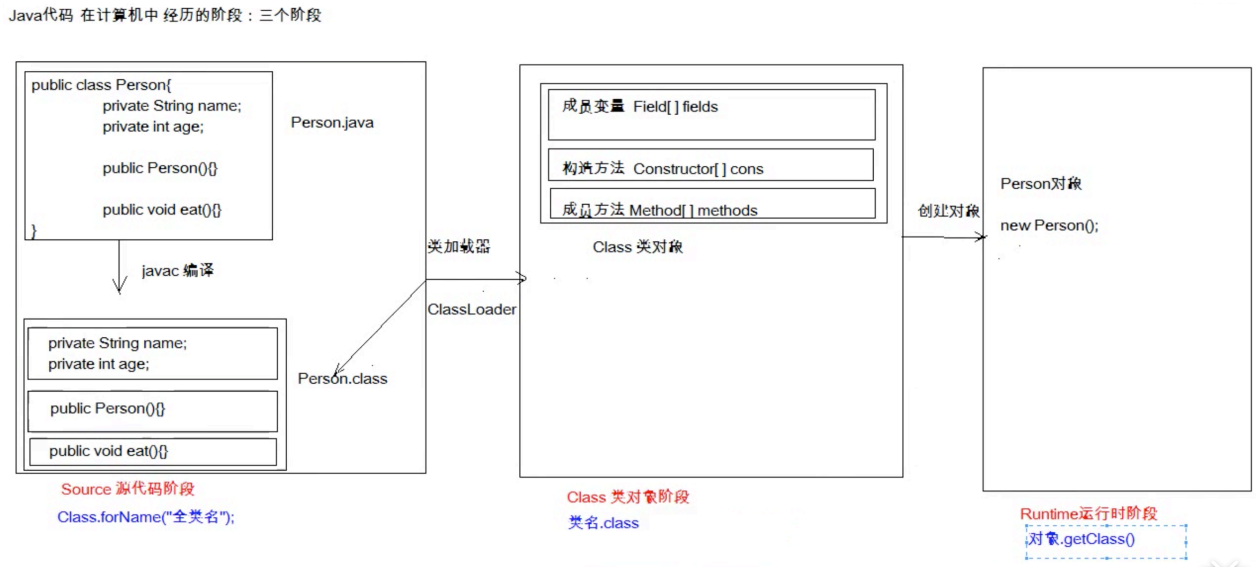
反射机制-入门

一、反射概述

1.反射:

- 框架设计的灵魂
- 将类的各个组成部分封装为其他对象，这就是反射机制
- 好处:
  - 可以在程序运行过程中，操作这些对象
  - 可以解耦，提高程序的可扩展性
- 改配置文件与改代码区别:
  - 改代码需要重新编译-》重新部署
  - 改配置文件、配置文件仅仅是物理文件，修改后不需要重新部署、还可以使得程序扩展性更强

2.框架:半成品软件，可以在框架的基础上进行软件开发，简化编码



二、反射API

1.获取class对象的方式:

- `Class.forName("全类名")`：
  - 将字节码文件加载进内存，返回class对象
  - 多用于配置文件，将类名定义在配置文件中，读取文件，加载类
- `类名.class`：
  - 通过类名的属性class获取
  - 多用于参数的传递
- `对象. getClass()`：

    - getClass( )方法在object类中定义着。
    - 多用于对象的获取字节码的方式
- 结论:
  同一个字节码文件(.class)在-次程序运行过程中，只会被加载-次, 不论通过哪- -种方式获取的Class对象都是同一个

```
/**
 * 获取类名
 */
public class ReflectDemo1 {
    public static void main(String[] args) throws ClassNotFoundException {
        //1.Class.forName("全类名")
        Class cls1 = Class.forName("cn.itcast.domain.Person");
        System.out.println(cls1);
        //2.类名.class
        Class cls2 = Person.class;
        System.out.println(cls2);
        //3.对象.getClass()
        Person p = new Person();
        Class cls3 = p.getClass();
        System.out.println(cls3);
        //== 比较三个对象
        System.out.println(cls1 == cls2);//true
        System.out.println(cls1 == cls3);//true
    }
}
```

2.Class对象功能:

获取功能:

1.获取成员变量们

```
Field[] getFields() //限定public
Field getField(string name)
Field[] getDeclaredFields() //全，但取private需要暴力反射
Field getDeclaredField(string name)
```

2.获取构造方法们

```
Constructor<?>[] getConstructors()
Constructor<T> getConstructor(类<?>... parameterTypes )
Constructor<T> getDeclaredConstructor(类<?>... parameterTypes)
Constructor<?>[] getDeclaredConstructors()
```

3.获取成员方法们:

```
Method[] getMethods() //限定public
Method getMethod(String name, 类<?>... parameterTypes)
Method[] getDeclaredMethods() //全，但取private需要暴力反射
Method getDeclaredMethod(String name, 类<?>... parameterTypes )
```

4.获取类名

```
string getName()
```

3.Field :成员变量:

1.设说值：`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`

```
/**
 * Class对象获取成员变量
 */
public class ReflectDemo2 {
    public static void main(String[] args) throws Exception {
        Class personClass = Person.class;
        //1. Field[] getFields( )获取所有public修饰的成员变量
        Field[] fields = personClass.getFields();
        for (Field field : fields) {
            System.out.println(field);
        }
        System.out.println("----------");
        //2.Field getField(String name)
        Field a = personClass.getField("a");
        //获取成员变量的值
        Person p = new Person();
        Object value = a.get(p);
        System.out.println(value);
        //设置a的值
        a.set(p, "张三");
        System.out.println(p);

        System.out.println("====================");
        //Field[] getDeclaredFields(): 获取所有的成员变址，不考虑修饰符
        Field[] declaredFields = personClass.getDeclaredFields();
        for (Field declaredField : declaredFields) {
            System.out.println(declaredField);
        }
        //Field getDeclaredField(String name)
        Field d = personClass.getDeclaredField("d");
        //忽略访问权限修饰符的安全检查->暴力反射
```

```
        d.setAccessible(true);
        Object value2 = d.get(p);
        System.out.println(value2);
    }
}
```

4.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

```
/**
 * Class API：Constructor构造对象
 */
public class ReflectDemo3 {
    public static void main(String[] args) throws Exception {
        Class personClass = Person.class;
        // Constructor API
        Constructor constructor = personClass.getConstructor(String.class,
int.class);
        System.out.println(constructor);
        //创建对象
        Object person = constructor.newInstance("张三", 23);
        System.out.println(person);
        System.out.println("----------");
        Constructor constructor1 = personClass.getConstructor();
        System.out.println(constructor1);
        //创建对象
        Object person1 = constructor1.newInstance();
        System.out.println(person1);
        //空参构造方法用Class自带(已经弃用)
        Object o = personClass.newInstance();
        System.out.println(o);
    }
}
```

5.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`

```
/**
 * Class API：Method对象
 */
public class ReflectDemo4 {
    public static void main(String[] args) throws Exception {
        Class personClass = Person.class;
```

```
        //获取指定名称的方法
        Method eatMethod = personClass.getMethod("eat");
        Person p = new Person();
        //执行方法
        eatMethod.invoke(p);
        //获得方法带参数
        Method eatMethod2 = personClass.getMethod("eat", String.class);
        //执行方法
        eatMethod2.invoke(p, "饭");
        System.out.println("---------------");
        //获取所有public修饰的方法
        Method[] methods = personClass.getMethods();
        for (Method method : methods) {
            System.out.println(method);
            String name = method.getName();
            System.out.println(name);
            //method.setAccessible(true);
        }
    }
}
```

三、反射案例

```
public class ReflectDemoFrameWork {
    public static void main(String[] args) throws Exception{
        //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1个集合
        //1.2.1获取class目录下的配置文件
        ClassLoader classLoader = ReflectDemoFrameWork.class.getClassLoader();
        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 cls = Class.forName(className);
        //4.创建对象
        Object obj = cls.newInstance();
        //5.获取方法对象
        Method method = cls.getMethod(methodName);
        //6.执行方法
        method.invoke(obj);
    }
}
```

三、反射案例

```
public class ReflectDemoFrameWork {
    public static void main(String[] args) throws Exception{
        //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1个集合
        //1.2.1获取class目录下的配置文件
        ClassLoader classLoader = ReflectDemoFrameWork.class.getClassLoader();
        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 cls = Class.forName(className);
        //4.创建对象
        Object obj = cls.newInstance();
        //5.获取方法对象
        Method method = cls.getMethod(methodName);
        //6.执行方法
        method.invoke(obj);
    }
}
```