

反射概述

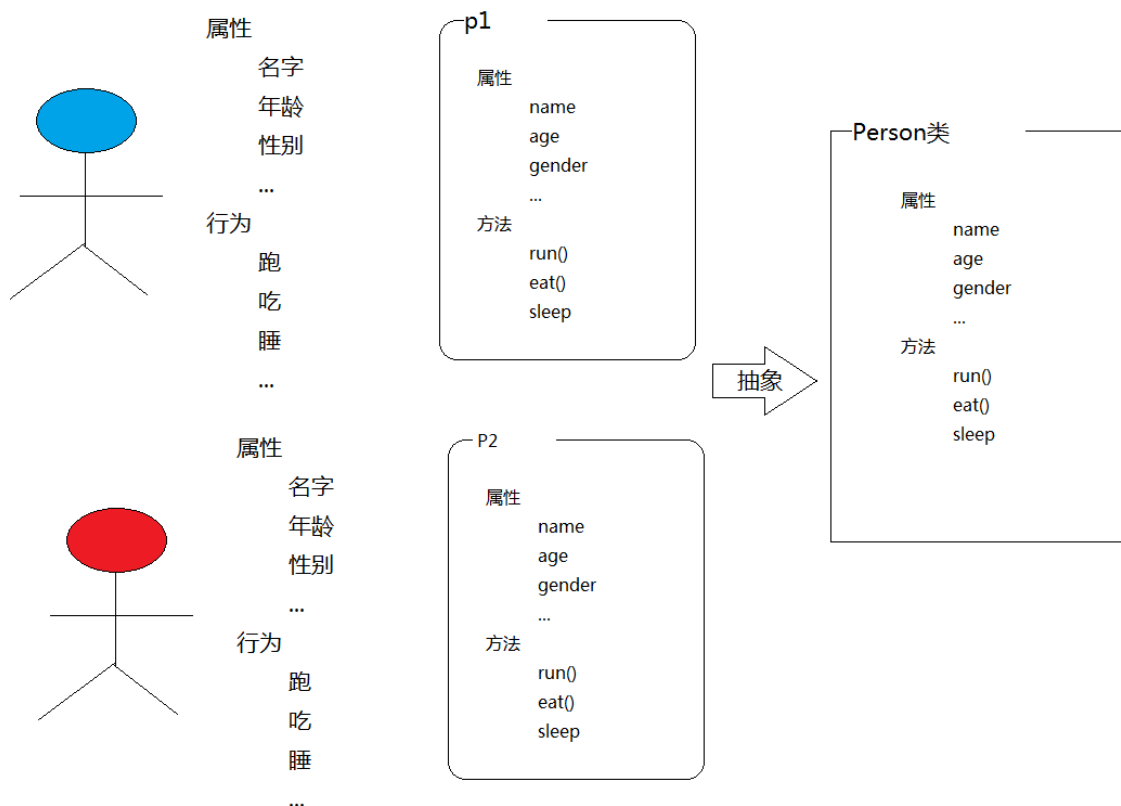
2018年8月28日 14:48

1. 反射概述

万物皆对象

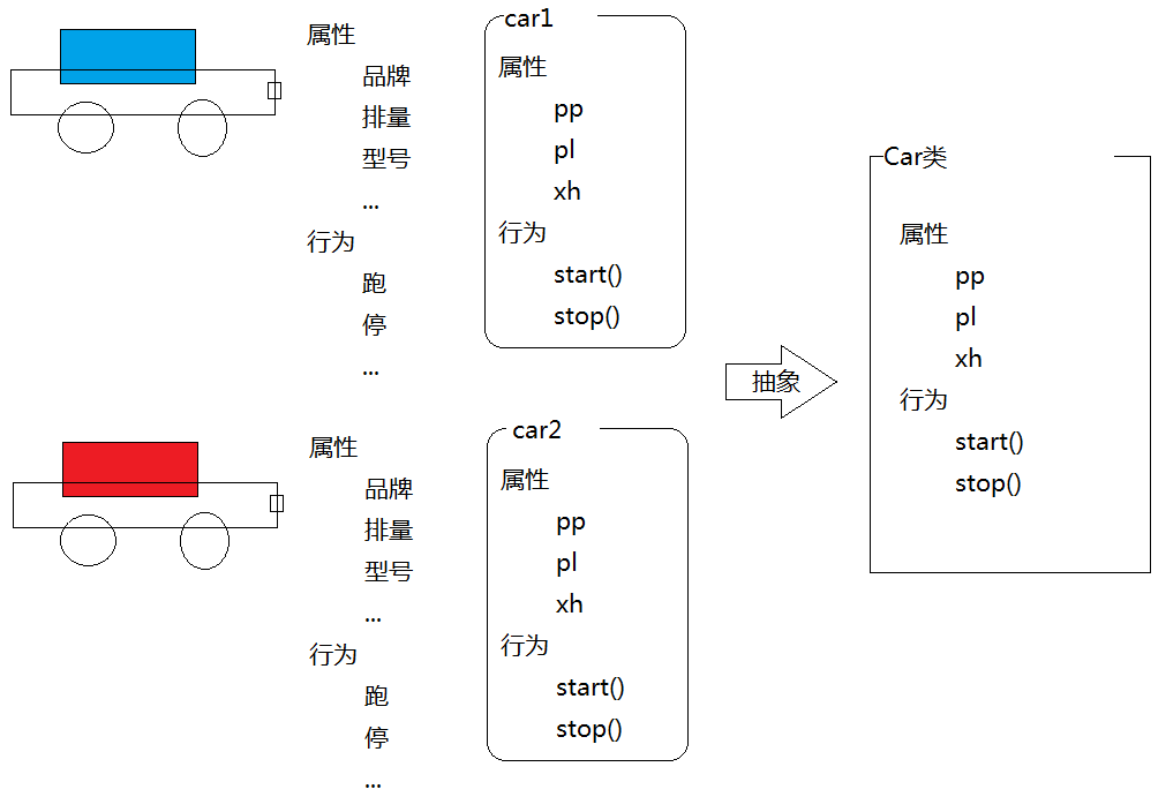
人是现实生活中真正存在的事物，在程序中用对象表示，经过抽象，可以得到代表人的类Person类

在程序中只要得到代表人的对象，就可以通过这个对象获知人相关属性，操作人相关的行为

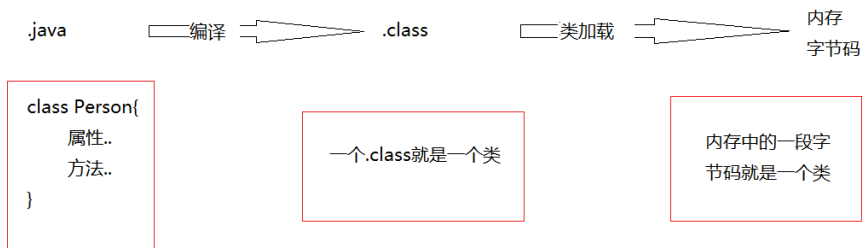


车是现实生活中真正存在的事物，在程序中用对象表示，经过抽象，可以得到代表车的类Car类

在程序中只要得到代表车的对象，就可以通过这个对象获知车相关属性，操作车相关的行为



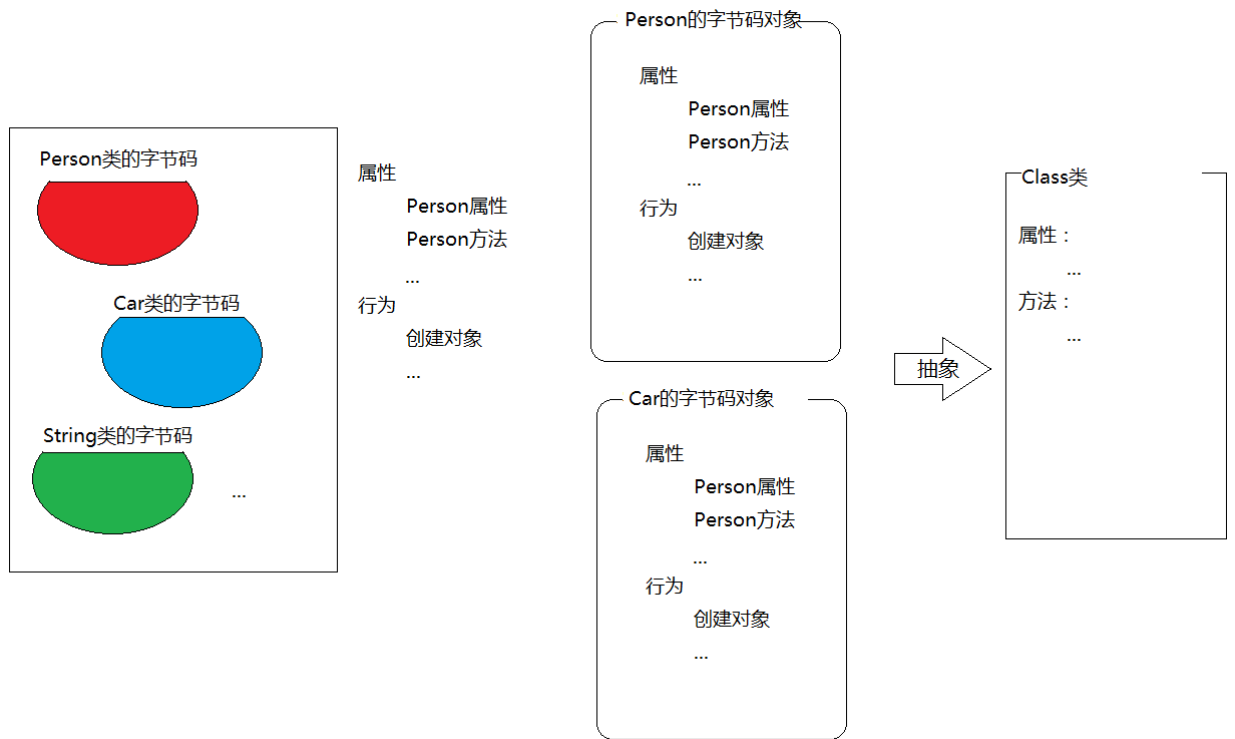
类是什么



**java.lang包下的类在虚拟机启动时立即加载
 **其他包下的类 在第一次使用时加载
 **类加载后一直驻留在内存中，为后续使用服务

字节码是现实生活中真正存在的事物，在程序中用对象表示，经过抽象，可以得到代表字节码的类Class类

在程序中只要得到代表字节码的对象，就可以通过这个对象获知字节码相关属性，操作字节码相关的行为



反射：

而所谓的反射，就是通过Class类作为入口，剖析字节码，获取字节码内部的信息 进行相关的操作

反射 - Class类

2018年8月28日 15:33

1. Class类概述

Class类是代表类(字节码)的类，是反射技术的入口，该类的对象代表内存中的字节码，通过获取该类的对象，可以剖析类(字节码)的属性 操作类(字节码)的行为，即实现反射。

Java.lang包下的类会在虚拟机启动时就加载，其他包下的类会在第一次使用时被加载，加载过后一直驻留在内存中，为后续访问服务，所以一个类的字节码，在内存中只有一份，所以一个类Class重复获取得到的也都是同一个对象。

2. 继承结构

java.lang

类 Class<T>

****Class类属于java.lang,不需要导包就可以使用**

****类上有泛型，用来配置当前Class类代表哪个类的字节码**

3. 获取Class

a. 通过 对象.getClass()获取 -- [有对象，用这个方法]

在Object类上提供了此方法，任意对象调用此方法都可以获取到该对象对应的类的Class对象

| | |
|-----------------------------|---|
| <code>Class<?></code> | <code>getClass()</code> 返回此 Object 的运行时类。 |
|-----------------------------|---|

案例：

```
Person p = new Person();  
Class clz = p.getClass();
```

b. 通过类名.class获取 -- [没对象，但是有类，用这个方法]

案例：

```
Person.class
```

c. 通过Class类提供的静态方法forName()传入类的全路径名来获取该类的Class对象 -- [即没有对象，也没有类，只有类的全路径名称是，用这个方法]

案例：

```
Class clz3 = Class.forName("cn.tedu.reflect.Person");
```

****Class.forName()方法** 将会去检查 当前虚拟机的内存中是否存在要获取的类的字节码，如果存在，则直接返回代表该字节码的Class对象，如果不存在，则加载对应类的.class文件到内存中变为字节码，再返回该字节码对应的Class对象

****同一个类获取多次Class得到的是一个对象**

案例：

```
package cn.tedu.reflect;

/**
 * 反射 - Class - 获取Class
 */
public class Demo01 {
    public static void main(String[] args) throws Exception {
        //1.通过对象的getClass()来获取
        Person p = new Person();
        Class clz1 = p.getClass();

        //2.类名.class来获取
        Class clz2 = Person.class;

        //3.通过Class.forName()来获取
        Class clz3 = Class.forName("cn.tedu.reflect.Person");

        //--同一个类获取多次Class得到的是一个对象
        System.out.println(clz1 == clz2);
        System.out.println(clz2 == clz3);
    }
}
```

4. Class类中的重要方法

重要方法：

| | |
|--|---|
| static Class <? > | forName(String className) 返回与带有给定字符串名的类或接口相关联的 Class 对象。 |
| I cast(Object obj) | 将一个对象强制转换成此 Class 对象所表示的类或接口。 |

| | |
|--|--|
| Class <?>[] | getInterfaces() 确定此对象所表示的类或接口实现的接口。 |
| Class <?> super I > | getSuperclass() 返回表示此 Class 所表示的实体（类、接口、基本类型或 void）的超类的 Class。 |
| Package | getPackage() 获取此类的包。 |
| String | getName() 以 String 的形式返回此 Class 对象所表示的实体（类、接口、数组类、基本类型或 void）名称。 |
| String | getSimpleName() 返回源代码中给出的底层类的简称。 |
| boolean | isArray() 判定此 Class 对象是否表示一个数组类。 |
| boolean | isInterface() 判定指定的 Class 对象是否表示一个接口类型。 |
| boolean | isEnum() 当且仅当该类声明为源代码中的枚举时返回 true。 |
| I | newInstance() 创建此 Class 对象所表示的类的一个新实例。 |

构造方法相关方法：

| | |
|---|--|
| Constructor <?>[] | getConstructors() 返回一个包含某些 Constructor 对象的数组，这些对象反映此 Class 对象所表示的类的所有公共构造方法。 |
| Constructor < I > | getConstructor(Class<?>... parameterTypes) 返回一个 Constructor 对象，它反映此 Class 对象所表示的类的指定公共构造方法。 |
| Constructor < I > | getDeclaredConstructor(Class<?>... parameterTypes) 返回一个 Constructor 对象，该对象反映此 Class 对象所表示的类或接口的指定构造方法。 |

| | |
|--------------------------------------|---|
| Constructor <?>[] | getDeclaredConstructors() 返回 Constructor 对象的一个数组，这些对象反映此 Class 对象表示的类声明的所有构造方法。 |
|--------------------------------------|---|

成员属性相关方法：

| | |
|--------------------------|--|
| Field [] | getFields() 返回一个包含某些 Field 对象的数组，这些对象反映此 Class 对象所表示的类或接口的所有可访问公共字段。 |
| Field | getField(String name) 返回一个 Field 对象，它反映此 Class 对象所表示的类或接口的指定公共成员字段。 |
| Field | getDeclaredField(String name) 返回一个 Field 对象，该对象反映此 Class 对象所表示的类或接口的指定已声明字段。 |
| Field [] | getDeclaredFields() 返回 Field 对象的一个数组，这些对象反映此 Class 对象所表示的类或接口所声明的所有字段。 |

成员方法相关的方法：

| | |
|--|--|
| Method [] | getMethods() 返回一个包含某些 Method 对象的数组，这些对象反映此 Class 对象所表示的类或接口（包括那些由该类或接口声明的以及从超类和超接口继承的那些的类或接口）的公共 <i>member</i> 方法。 |
| Method d | getMethod(String name, Class<?>... parameterTypes) 返回一个 Method 对象，它反映此 Class 对象所表示的类或接口的指定公共成员方法。 |
| Method d | getDeclaredMethod(String name, Class<?>... parameterTypes) 返回一个 Method 对象，该对象反映此 Class 对象所表示的类或接口的指定已声明方法。 |
| Method d [] | getDeclaredMethods() 返回 Method 对象的一个数组，这些对象反映此 Class 对象表示的类或接口声明的所有方法，包括公共、保护、默认（包）访问和私有 |

方法，但不包括继承的方法。

****没有Declared的方法 只能获取公有的，有Declared方法可以获取所有的，但是虽然可以获取所有的，但不可访问的，仍然不可访问。但是可以通过调用 setAccessible(true)强制打开访问权限，这种方式是反射独有的突破java的访问权限控制的能力，很强大，但一定要慎用**

5. 案例

```
package cn.tedu.reflect;

/**
 * Class类的普通方法
 */
public class Demo02 {
    public static void main(String[] args) throws Exception {
        //1.cast()
        Object obj = new Person("zs",19);
        Person p1 = (Person) obj;
        Person p2 = Person.class.cast(obj);

        //2.getInterfaces()
        Class<?>[] intfs = Person.class.getInterfaces();
        for(Class intf : intfs){
            System.out.println(intf.getName());
        }

        //3.getSuperclass()
        System.out.println(Person.class.getSuperclass());

        //4.getPackage()
        System.out.println(Person.class.getPackage().getName());

        //5.getName() getSimpleName()
        System.out.println(Person.class.getName());
        System.out.println(Person.class.getSimpleName());

        //6.newInstance()
        Person p = Person.class.newInstance();
        p.sleep();
    }
}
```


反射 - Constructor类

2018年8月28日 16:11

1. Constructor概述

代表类的构造方法的类

2. Constructor的获取

在Class类中提供了方法可以用来获取Class类代表的类的构造方法

Class类中提供：

| | |
|---------------------------------------|--|
| Constructor <?> [] | getConstructors() 返回一个包含某些 Constructor 对象的数组，这些对象反映此 Class 对象所表示的类的所有公共构造方法。 |
| Constructor <I> | getConstructor (Class <?>... parameterTypes) 返回一个 Constructor 对象，它反映此 Class 对象所表示的类的指定公共构造方法。 |
| Constructor <I> | getDeclaredConstructor (Class <?>... parameterTypes) 返回一个 Constructor 对象，该对象反映此 Class 对象所表示的类或接口的指定构造方法。 |
| Constructor <?> [] | getDeclaredConstructors() 返回 Constructor 对象的一个数组，这些对象反映此 Class 对象表示的类声明的所有构造方法。 |

3. Constructor的使用

java.lang.reflect

类 **Constructor<T>**

| | |
|---------------------------------|---|
| String | getName() 以字符串形式返回此构造方法的名称。 |
| Class <?> [] | getParameterTypes() 按照声明顺序返回一组 Class 对象，这些对象表示此 Constructor 对象所表示构造方法的形参类型。 |
| I | newInstance (Object ... initargs) 使用此 Constructor 对象表示的构造方法来创建该构造方法的声明类的新实例，并用指定的初始化参数初始化该实例。 |

| | |
|------|--|
| void | <u>setAccessible</u> (boolean flag) 将此对象的 accessible 标志设置为指示的布尔值。 |
|------|--|

4. 案例：

```

package cn.tedu.reflect;

import java.lang.reflect.Constructor;

/**
 * 反射 - Class - 构造方法 Constructor
 */
public class Demo03 {
    public static void main(String[] args) throws Exception {
        //1.获取该类的所有的公有构造方法
        //    Constructor[] cs = Person.class.getConstructors();

        //2.获取指定构造方法
        //    Constructor<Person> cons1 = Person.class.getConstructor();
        //    Constructor<Person> cons2 = Person.class.getConstructor(String.class,int.class);

        //3.用构造方法创建对象
        //    Person p = cons2.newInstance("zs",19);
        //    p.sleep();

        //4.访问私有构造方法
        Constructor<Person> constructor = Person.class.getDeclaredConstructor(String.class);
        constructor.setAccessible(true);
        Person p = constructor.newInstance("ww");
        System.out.println(p.name);
    }
}

```

反射 - Fileds类

2018年8月28日 16:40

1. Fields概述

代表类的成员属性的类

2. Fields的获取

在Class类中提供了方法，获取Class代表的类的成员属性的对象

Class类上提供的方法：

| | |
|-------------------------|--|
| Field[] | getFields() 返回一个包含某些 Field 对象的数组，这些对象反映此 Class 对象所表示的类或接口的所有可访问公共字段。 |
| Field | getField(String name) 返回一个 Field 对象，它反映此 Class 对象所表示的类或接口的指定公共成员字段。 |
| Field | getDeclaredField(String name) 返回一个 Field 对象，该对象反映此 Class 对象所表示的类或接口的指定已声明字段。 |
| Field[] | getDeclaredFields() 返回 Field 对象的一个数组，这些对象反映此 Class 对象所表示的类或接口所声明的所有字段。 |

3. Field类提供的方法

| | |
|--------------------------------|---|
| String | getName() 返回此 Field 对象表示的字段名称。 |
| Class<?> | getType() 返回一个 Class 对象，它标识了此 Field 对象所表示字段的声明类型。 |

| | |
|-------------------------|--|
| Object | get(Object obj) 返回指定对象上此 Field 表示的字段值。 |
| boolean | getBoolean(Object obj) 获取一个静态或实例 boolean 字段的值。 |

| | |
|--------|---|
| byte | getBytes(Object obj) 获取一个静态或实例 byte 字段的值。 |
| char | getChar(Object obj) 获取 char 类型或另一个通过扩展转换可以转换为 char 类型的基本类型的静态或实例字段的值。 |
| double | getDouble(Object obj) 获取 double 类型或另一个通过扩展转换可以转换为 double 类型的基本类型的静态或实例字段的值。 |
| float | getFloat(Object obj) 获取 float 类型或另一个通过扩展转换可以转换为 float 类型的基本类型的静态或实例字段的值。 |
| int | getInt(Object obj) 获取 int 类型或另一个通过扩展转换可以转换为 int 类型的基本类型的静态或实例字段的值。 |
| long | getLong(Object obj) 获取 long 类型或另一个通过扩展转换可以转换为 long 类型的基本类型的静态或实例字段的值。 |
| short | getShort(Object obj) 获取 short 类型或另一个通过扩展转换可以转换为 short 类型的基本类型的静态或实例字段的值。 |

| | |
|------|--|
| void | set(Object obj, Object value) 将指定对象变量上此 Field 对象表示的字段设置为指定的新值。 |
| void | setBoolean(Object obj, boolean z) 将字段的值设置为指定对象上的一个 boolean 值。 |
| void | setByte(Object obj, byte b) 将字段的值设置为指定对象上的一个 byte 值。 |
| void | setChar(Object obj, char c) 将字段的值设置为指定对象上的一个 char 值。 |
| void | setDouble(Object obj, double d) 将字段的值设置为指定对象上的一个 double 值。 |
| void | setFloat(Object obj, float f) |

| | |
|------|---|
| | 将字段的值设置为指定对象上的一个 float 值。 |
| void | <code>setInt(Object obj, int i)</code> 将字段的值设置为指定对象上的一个 int 值。 |
| void | <code>setLong(Object obj, long l)</code> 将字段的值设置为指定对象上的一个 long 值。 |
| void | <code>setShort(Object obj, short s)</code> 将字段的值设置为指定对象上的一个 short 值。 |

| | |
|------|---|
| void | <code>setAccessible(boolean flag)</code> 将此对象的 accessible 标志设置为指示的布尔值。 |
|------|---|

4. 案例：

```
package cn.tedu.reflect;

import java.lang.reflect.Field;

/**
 * 反射 - Class - 成员属性 Fields
 */
public class Demo04 {
    public static void main(String[] args) throws Exception {
        //1.获取Person类的所有成员属性
        //    Field[] fields = Person.class.getFields();
        //2.获取Person类的指定成员属性
        //    Field field = Person.class.getField("name");
        //3.获取属性名称
        //    Field field = Person.class.getField("name");
        //    System.out.println(field.getName());
        //4.获取属性的类型
        //    Field field = Person.class.getField("name");
        //    System.out.println(field.getType());
        //5.获取属性值
        //    Person p = new Person("zs",19);
        //    Field field = Person.class.getField("name");
        //    String name = (String) field.get(p);
        //    System.out.println(name);
        //6.设置属性值
        //    Person p = new Person("zs",19);
        //    Field field = Person.class.getField("name");
        //    field.set(p, "ls");
    }
}
```

```
//      System.out.println(p.name);

//7.访问私有属性
Person p = new Person("zs",19);
Field field = Person.class.getDeclaredField("age");
field.setAccessible(true);
int age = (int) field.get(p);
System.out.println(age);
    }
}
```

反射 - Method类

2018年8月28日 16:51

1. Method概述

代表类的成员方法的类

2. Method对象获取

在Class类上提供了获取Class代表的字节码中的成员方法的方法

Class类上提供的方法：

| | |
|--------------------------|--|
| Method[] | getMethods() 返回一个包含某些 Method 对象的数组，这些对象反映此 Class 对象所表示的类或接口（包括那些由该类或接口声明的以及从超类和超接口继承的那些的类或接口）的公共 <i>member</i> 方法。 |
| Method | getMethod(String name, Class<?>... parameterTypes) 返回一个 Method 对象，它反映此 Class 对象所表示的类或接口的指定公共成员方法。 |
| Method | getDeclaredMethod(String name, Class<?>... parameterTypes) 返回一个 Method 对象，该对象反映此 Class 对象所表示的类或接口的指定已声明方法。 |
| Method[] | getDeclaredMethods() 返回 Method 对象的一个数组，这些对象反映此 Class 对象表示的类或接口声明的所有方法，包括公共、保护、默认（包）访问和私有方法，但不包括继承的方法。 |

3. Method提供的方法

| | |
|--|---|
| String | getName() 以 String 形式返回此 Method 对象表示的方法名称。 |
| Class<?> | getReturnType() 返回一个 Class 对象，该对象描述了此 Method 对象所表示的方法的正式返回类型。 |
| TypeVariable<Method>[] | getTypeParameters() 返回 TypeVariable 对象的数组，这些对象描述了由 GenericDeclaration 对象表示的一般声明按声明顺序来声明的类型变量。 |
| Object | invoke(Object obj, Object... args) 对带有指定参数的指定对象调用由此 Method 对象表示的底层方法。 |

void

[**setAccessible**](#)(boolean flag)

将此对象的 accessible 标志设置为指示的布尔值。

4. 案例

```
package cn.tedu.reflect;

import java.lang.reflect.Method;

/**
 * 反射 - Class - 成员方法 Method
 */
public class Demo05 {
    public static void main(String[] args) throws Exception {
        //1.获取类上所有的成员方法
        // Method[] methods = Person.class.getMethods();
        //2.获取类上指定的成员方法
        // Method method = Person.class.getMethod("eat", String.class,int.class);
        //3.获取方法名 获取返回值类型
        // Method[] methods = Person.class.getMethods();
        // for(Method method : methods){
        //     System.out.println(method.getName());
        //     System.out.println(method.getReturnType());
        //     Class<?>[] parameterTypes = method.getParameterTypes();
        //     for(Class pt : parameterTypes){
        //         System.out.println(pt.getName());
        //     }
        //     System.out.println("----");
        // }
        //4.执行方法
        // Person p = new Person("zs",19);
        // Method method = Person.class.getMethod("eat", String.class,int.class);
        // method.invoke(p, "羊肉串",10);

        //5.操作私有方法
        Person p = new Person("zs",19);
        Method method = Person.class.getDeclaredMethod("study", null);
        method.setAccessible(true);
        method.invoke(p, null);
    }
}
```


反射的应用

2018年8月29日 9:18

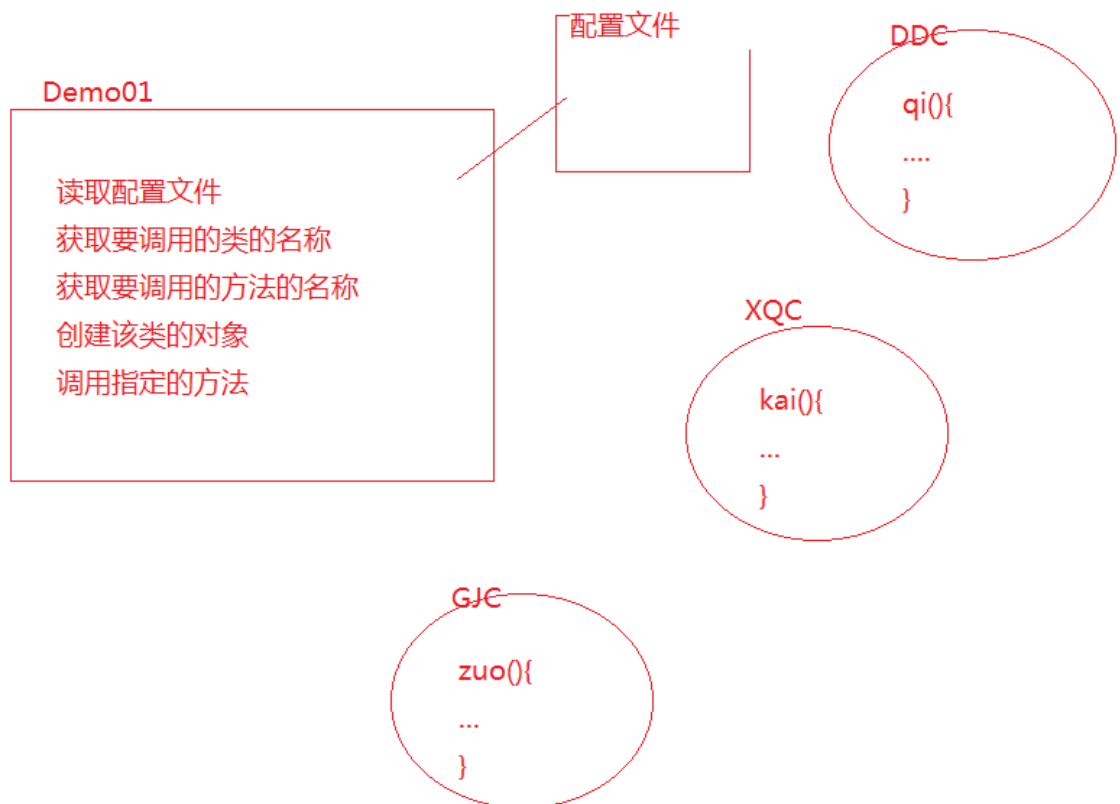
1. 反射的应用场景

通常的代码中是用不上反射，一般只在框架级别的代码中才会用到反射

所谓的框架，其实就是程序的架子，将程序开发过程中通用的部分提取出来，形成程序的架子，此后再需要开发类似的程序时，不需要重头开发，而是基于这个程序的架子，在其中编写不一样的部分就可以了。

框架中经常要提供很好的灵活性，可以根据相关的配置来调用未来传入框架的类或方法，而在编写框架时，根本不知道未来传入的类或方法是那些，此时无法直接new对象，也无法直接调用方法，此时就可以使用反射技术来，反射创建对象，操作属性和方法。

案例：出门根据配置文件选择交通工具的案例



```
package cn.tedu.reflect;
```

```
public class DDC {
```

```
    public void 骑(){
```

```
        System.out.println("骑着我的雅迪电动车，风驰电掣~~");
```

```
    }
```

```

}
package cn.tedu.reflect;

public class GJC {
    public void 坐(){
        System.out.println("坐着我的大公交，4毛钱随便坐~~");
    }
}

package cn.tedu.reflect;

public class XQC {
    public void 开(){
        System.out.println("开着我的小奥拓，大马路上飙车~~");
    }
}

package cn.tedu.reflect;

public class FJ {
    public void 飞(){
        System.out.println("坐上我的直升机，想飞哪里飞哪里~~~");
    }
}

package cn.tedu.reflect;

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

public class Demo01 {
    public static void main(String[] args) throws Exception {
        System.out.println("出门。。。选择交通工具。。。");

        //此处根据配置文件 选择出行工具和方法
        InputStream in = new FileInputStream("demo01.properties");
        Properties prop = new Properties();
        prop.load(in);
        in.close();

        String clzName = prop.getProperty("className");
        String mName = prop.getProperty("methodName");
    }
}

```

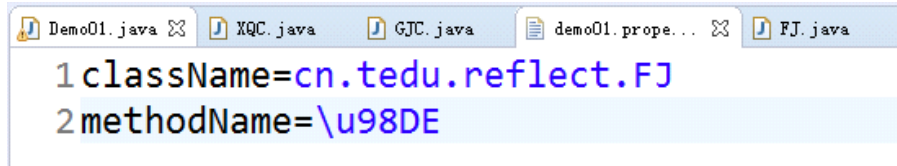
```
Class clz = Class.forName(clzName);  
Object obj = clz.newInstance();  
Method method = clz.getMethod(mName, null);  
method.invoke(obj, null);
```

```
System.out.println("到达目的地。。。");
```

```
}
```

```
}
```

Demo01可以根据配置文件，决定创建哪个类调用哪个方法



```
1 className=cn.tedu.reflect.FJ  
2 methodName=\u98DE
```