类的加载

2018年7月30日 15:50

类的加载过程：
1.加载　将class文件读入内存，并创建一个class对象
2.连接
3.初始化

类加载器：
1.根类加载器
2.扩展加载器
3.类加载器

java反射机制：
在运行状态，对于任意一个类，都能知道这个类的所有属性和方法
通过class文件对象，去使用该文件中的成员变量，构造方法和成员方法
class类：
成员变量　Field
构造方法　Construtor
构造方法　Method

获取class文件对象

2018年7月31日 17:35

```
/*
 * 获取class文件对象的方式
 * 1.Object类的getClass()方法
 * 2.数据类型的静态属性class
 * 3.Class类中的静态方法：
 *                          public static Class forName(String className)
 *                          带全路径
 * 自己用：第二种，方便
 * 开发：第三种，字符串可以配置到配置文件中
 */
public class ReflectDemo {
	public static void main(String[] args) throws ClassNotFoundException {
		//方式1
		Person p = new Person();
		Class c = p.getClass();

		Person p2 = new Person();
		Class c2 = p2.getClass();

		System.out.println(p == p2);// false
		System.out.println(c == c2);// true

		//方式2
		Class c3 = Person.class;

		//方式3
		Class c4=Class.forName("cn.itcast_01.Person");
		System.out.println(c==c4);
	}
}
```

反射获得无参构造

2018年7月31日 17:35

```
/*
 * 通过反射获取构造方法并使用
 */
public class ReflectDemo {
	public static void main(String[] args) throws Exception {
		// 获取字节码文件对象
		Class c = Class.forName("cn.itcast_01.Person");

		// 获取构造方法
		// public Constructor[] getConstructors()：所有公共构造方法
		// public Constructor[] getDeclaredConstructors:所有构造方法
		// Constructor[] cons =c.getConstructors();
		Constructor[] cons = c.getDeclaredConstructors();
		for (Constructor con : cons) {
			System.out.println(con);
		}

		// 获取单个构造方法
		Constructor con = c.getConstructor();
		Object obj = con.newInstance();
		System.out.println(obj);
	}
}
```

反射获取带参构造

2018年7月31日 17:53

```
/*
 * 通过反射获取带参构造并使用
 */
public class ReflectDemo2 {
	public static void main(String[] args) throws Exception{
		Class c = Class.forName("cn.itcast_01.Person");

		//获取带参构造器
		Constructor con = c.getConstructor(String.class,int.class,String.class);

		//创建对象
		Object obj = con.newInstance("椰子皮",22,"沈阳");
		System.out.println(obj);
	}
}
```

反射获取私有构造

2018年7月31日 20:50

```
/*
 * 通过反射获取私有构造方法并使用
 */
public class ReflectDemo3 {
	public static void main(String[] args) throws Exception {
		Class c = Class.forName("cn.itcast_01.Person");

		// NoSuchMethodException
		// IllegalAccessException
		Constructor con = c.getDeclaredConstructor(String.class);

		// 暴力访问
		con.setAccessible(true);// 取消java语言访问检查

		Object obj = con.newInstance("椰子皮");

		System.out.println(obj);
	}
}
```

反射获取成员变量

2018年7月31日 21:10

```
/*
 * 通过反射获取成员变量并使用
 */
public class ReflectDemo {
	public static void main(String[] args) throws Exception {
		Class c = Class.forName("cn.itcast_01.Person");
		// 获取所有成员变量
		Field[] fields = c.getDeclaredFields();

		for (Field field : fields) {
			System.out.println(field);
		}

		Constructor con = c.getConstructor();
		Object obj = con.newInstance();

		// 获取单个成员变量
		Field addressField = c.getField("address");
		// 给对象赋值 public void set(Object obj,Object value)
		addressField.set(obj, "沈阳");
		System.out.println(obj);

		// 获取单个私有成员变量
		Field nameField = c.getDeclaredField("name");
		nameField.setAccessible(true);
		nameField.set(obj, "椰子皮");
		System.out.println(obj);

	}
}
```

反射获取方法

2018年7月31日 21:41

```
public class ReflectDemo {
	public static void main(String[] args) throws Exception {
		Class c = Class.forName("cn.itcast_01.Person");

		// 获取所有方法
		// Method[] methods = c.getMethods();//获取包括父类的所有方法
		Method[] methods = c.getDeclaredMethods();// 获取自己的方法

		// 获取单个方法
		Constructor con = c.getConstructor();
		Object obj = con.newInstance();

		// 获取单个方法
		Method m1 = c.getMethod("show");
		// invoke
		m1.invoke(obj);
	}
}
```

反射配置文件

2018年7月31日 22:48

```
/*
 * 通过配置文件运行类中的方法
 * 反射：
 * 需要有配置文件配合使用
 * 用class.txt代替
 * 两个键：className  methodName
 */
public class Test {
	public static void main(String[] args) throws Exception{
		//加载键值对
		Properties prop = new Properties();
		FileReader fr = new FileReader("class.txt");
		prop.load(fr);
		fr.close();

		//获取数据
		String className =prop.getProperty("className");
		String methodName = prop.getProperty("methodName");

		Class c = Class.forName(className);
		Constructor con = c.getConstructor();
		Object obj = con.newInstance();

		Method m = c.getMethod(methodName);
		m.invoke(obj);
	}
}
```

配置文件class.txt

```
className=cn.itcast_05.Teacher
methodName=show
```

动态代理

2018年7月31日 22:49

代理
动态代理：在程序运行过程中产生的对象

Proxy.newProxyInstance
生成代理对象

模板设计模式

2018年8月1日 10:20

定义一个算法的骨架，并在其子类编写

```
public abstract class GetTime {
    // 给出一段代码的运行时间
    public long getTime() {
        long start = System.currentTimeMillis();

        code();

        long end = System.currentTimeMillis();
        return end - start;

    }

    public abstract void code();

}
```

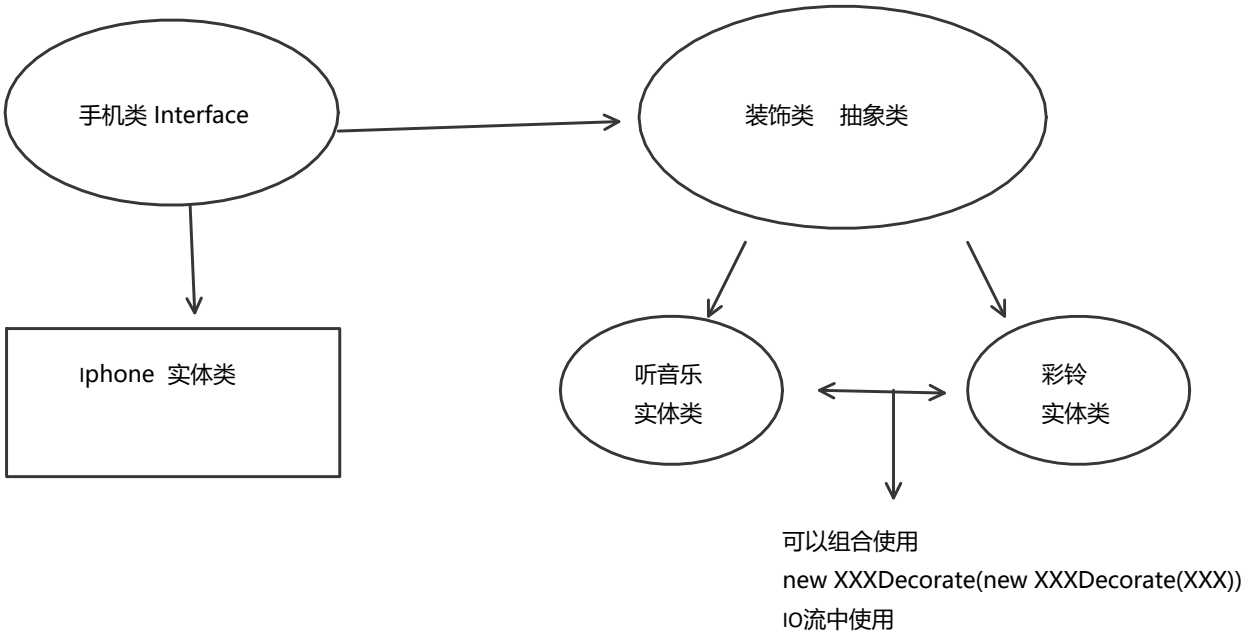
```
public class ForDemo extends GetTime {

    @Override
    public void code() {
        for (int x = 0; x < 10000; x++) {
            System.out.println(x);
        }
    }
}
```

```
public class GetTimeDemo {
    public static void main(String[] args) {
        GetTime gt = new ForDemo();
        System.out.println(gt.getTime());
    }
}
```

装饰模式

2018年8月1日 10:37



可以组合使用

new XXXDecorate(new XXXDecorate(XXX))

IO流中使用

JDK5的新特性

2018年8月1日 10:53

1.自动装箱与拆箱
2.泛型
3.增强for
4.静态导入
5.可变参数
6.枚举

自己实现枚举类

2018年8月1日 11:11

```
public class Direction {
    //构造私有，避免无限创建
    private Direction() {}

    public static final Direction FRONT = new Direction();
    public static final Direction BEHIND = new Direction();
    public static final Direction RIGHT = new Direction();
    public static final Direction LEFT = new Direction();
}
```

```
public class Direction2 {
    public static final Direction2 FRONT = new Direction2("前");
    public static final Direction2 BEHIND = new Direction2("后");
    public static final Direction2 RIGHT = new Direction2("左");
    public static final Direction2 LEFT = new Direction2("右");

    private String name;

    private Direction2(String name) {
        this.name=name;
    }

    public String getName() {
        return name;
    }
}
```

```
public abstract class Direction3 {
    //匿名内部类
    public static final Direction3 FRONT = new Direction3("前") {
        public void show() {
            System.out.println("前");
        }
    };
    public static final Direction3 BEHIND = new Direction3("后"){
        public void show() {
            System.out.println("后");
        }
    };;
    public static final Direction3 RIGHT = new Direction3("左"){
        public void show() {
            System.out.println("左");
        }
    };;
    public static final Direction3 LEFT = new Direction3("右"){
```

```
public class DirectionDemo {
    public static void main(String[] args) {
        Direction d = Direction.FRONT;
        System.out.println(d);//cn.itcast_08.Direction@
        70dea4e
        System.out.println("------");

        Direction2 d2 = Direction2.FRONT;//cn.itcast_
        08.Direction2@5c647e05
        System.out.println(d2);
        System.out.println(d2.getName());//前
        System.out.println("------");

        Direction3 d3 = Direction3.FRONT;
        System.out.println(d3);
        System.out.println(d3.getName());
        d3.show();
    }
}
```

```
        public void show() {
            System.out.println("右");
        }
    };;

    private String name;

    private Direction3(String name) {
        this.name=name;
    }

    public String getName() {
        return name;
    }

    //加入抽象方法
    public abstract void show();
}
```

enum实现枚举类

2018年8月1日 11:32

```
/*
 * 通过JDK5以后的枚举类来作枚举
 */
public enum Direction {
	FRONT,BEHIND,LEFT,RIGHT;
}
```

```
public enum Direction2 {
	FRONT("前"), BEHIND("后"), LEFT("左"), RIGHT("右");
	private String name;

	private Direction2(String name) {
		this.name = name;
	}

	public String getName() {
		return name;
	}
}
```

枚举类注意事项

2018年8月1日 11:42

1.枚举类必须放在最前面
2.枚举类后面没有东西，分号可以省略
3.枚举类可以有构造器，但必须是private的
4.枚举类可以有构造方法，必须重写该方法
5.枚举可在**Switch**中使用

常见方法：
int compareTo (E o)

String name()

int ordinal

to String()　自动重写

Valueof

values()　返回对象数组

JDK7新特性

2018年8月1日 11:59

1.二进制字面量

```
int x = 0b100101;
System.out.println(x);//37
```

2.数字字面量可以出现下划线

```
int y= 1_3100_1000;
System.out.println(y);
```

3.switch语句可以使用字符串

4.泛型简化

```
ArrayList<String> array = new ArrayList<>();
```

5.异常多个catch的合并

6.try-with-resourse语句

```
private static void method() {
		// try-with-resourse语句
		// 括号中的自动关闭
		try (FileReader fr = new FileReader("a.txt");
				FileWriter fw = new FileWriter("b.txt");) {
			int ch = 0;
			while ((ch = fr.read()) != -1) {
				fw.write(ch);
			}
		} catch (IOException e) {
			e.printStackTrace();
		}
	}
```