JDK8 新特性

2.3 节之前的内容请观看：https://www.wkcto.com/course/112

2.3 接口中默认方法

接口中方法默认使用 public abstract 修饰，接口中字段默认使用 public static final 修饰.

在 JDK8 中对接口进行增强，可以定义 default 修饰的方法,也可以定义 static 修饰的方法

default 修饰的方法,和 static 修饰的方法主要用于接口功能增强时,如果接口已经定义完成,并且也有若干的实现类实现了该接口.根据业务需求,需要在接口中再增强其他的功能，后面增强的功能可以使用 default 修饰. 之前定义好的实现类就不需要再进行修改

```
package com.wkcto.interfaces;

/**
 * Author：动力节点老崔
 */
public interface MyInterface {
    void m1();          //默认的抽象方法，需要在实现类中重写
    int XX = 121;      //字段默认 public static final 修饰

    //如果方法使用 default 修饰，表示该方法可以有默认的方法体,在实现类中可以重写,也可以不重写
    default    void    dm(){
```

```
        System.out.println("接口中使用 default 修饰的方法");
    }
    //静态方法
    static   void   sm(){
        System.out.println("接口中可以使用 static 定义静态方法");
    }
}
```

2.4 扩展注解支持

1 增加了 FunctionalInterface 注解

函数式接口注解,为 Labmda 表达式准备的,

当接口中只有一个方法是抽象方法时,则该接口可以声明为函数式接口

```
package com.wkcto.annotaitons;

/**
 * 函数式接口
 * Author : 动力节点老崔
 */
@FunctionalInterface                    //注解声明接口为函数式接口
public interface MyInterface2 {
    void   m1();

    default void dm(){
        System.out.println("default 修饰的方法有默认方法体");
    }
```

```
}
```

```
package com.wkcto.annotaitons;

/**
 * 函数式接口可以使用 Labmda 表达式
 * Author : 动力节点老崔
 */
public class Test01 {
    public static void main(String[] args) {
        //接口赋值匿名内部类对象
        MyInterface2  mi2 = new MyInterface2() {
            @Override
            public void m1() {
                System.out.println("在匿名内部类中重写接口的抽象方法");
            }
        };
        mi2.m1();          //执行匿名内部类对象的方法

        //MyInteface2 接口声明为了函数式接口,可以直接给接口引用赋值 Labmda 表达式
        mi2 = () -> System.out.println("给接口引用赋值 Lambda 表达式");
        mi2.m1();

    }
}
```

2 对元注解的增强

JDK8 扩展了注解的使用范围,在 ElementType 枚举类型中增强了两个枚举值:

ElementType.PARAMETER,表示注解能写在类型变量的声明语句中

ElementType.USE，表示注解能写在使用类型的任何语句中

增加了 Repeatable 元注解.JDK8 前的版本中,同一个注解在同一个位置只能使用一次,不能使用多次. 在 JDK8 中引入了重复注解,表示一个注解在同一个位置可以重复使用多次.

```
package com.wkcto.annotaitons;

import java.lang.annotation.*;

/**自定义可重复的注解

 * Author： 动力节点老崔
 */
@Repeatable(RoleAnnotions.class)
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE,ElementType.METHOD})
public @interface MyAnnotation {

    String role();          //定义角色属性

}
```

```
package com.wkcto.annotaitons;

import java.lang.annotation.ElementType;
import java.lang.annotation.Retention;
import java.lang.annotation.RetentionPolicy;
import java.lang.annotation.Target;

/**
 * 定义一个容器,可以包含若干的 MyAnnotation 注解

 * Author： 动力节点老崔

 */
```

```
@Retention(RetentionPolicy.RUNTIME)
@Target({ElementType.TYPE,ElementType.METHOD})
public @interface RoleAnnotions {
    MyAnnotation[] value();
}
```

```
package com.wkcto.annotaitons;

/**
 * 使用重复注解修饰一个类
 * Author : 动力节点老崔
 */
@MyAnnotation( role = "Husband")
@MyAnnotation(role = "Son")
@MyAnnotation(role = "Father")
public class Person {
}
```

```
package com.wkcto.annotaitons;

/**
 * 通过反射读取可重复注解的信息
 * Author : 动力节点老崔
 */
public class Test02 {
    public static void main(String[] args) {
        //创建 Class 对象
        Class<?> claxx = Person.class;

        //获得指定的自定义注解
        MyAnnotation[] myAnnotations = claxx.getDeclaredAnnotationsByType(MyAnnotation.class);
        //遍历数组,打印角色属性
        for (MyAnnotation annotation : myAnnotations){
            System.out.println( annotation.role());
```

```
        }
    }
}
```

2.5 Optional 类

JDK8 中引入的 Optional 类可以解决空指针异常，让我们省略繁琐的非空判断.

Optional 类就是一个可以为 null 容器，或者保存指定类型的数据，或者为 null.

static <T> Optional<T>	empty() 返回一个空的 Optional 对象
boolean	equals(Object obj)
Optional<T>	filter(Predicate<? super T> predicate) 如果有值,返回符合 predicate 条件的 Optional 对象，否则返回空的 Optional 对象
<U> Optional<U>	flatMap(Function<? super T,Optional<U>> mapper) 如果有值,执行 mapper 映射函数,返回 Optional 对象，如果没有值返回空的 Optional 对象
T	get() 如果值存在返回，如果不存在抛出异常 NoSuchElementException.

int	hashCode()
void	ifPresent(Consumer<? super T> consumer) 如果值存在就执行 consumer 函数,否则什么也不做
boolean	isPresent() 判断值是否存在
<U> Optional<U>	map(Function<? super T,? extends U> mapper) 如果值存在就执行 mapper 映射函数,
static <T> Optional<T>	of(T value) 把指定的 value 值封装为 Optional 对象,如果 value 为 null,产生空指针异常
static <T> Optional<T>	ofNullable(T value) 把指定的 value 值封装为 Optional 对象,如果 value 为 null 返回一个空的 Optional 对象
T	orElse(T other) 如果值存在就返回,如果不存在返回 other
T	orElseGet(Supplier<? extends T> other) 如果存在就返回值,如果不存在,执行 Supplier 返回另外一个值
<X extends Throwable> T	orElseThrow(Supplier<? extends X> exceptionSupplier) 如果存在就返回该值,如果不存在抛出由 exceptionSupplier 生成的异常
String	toString()

```
package com.wkcto.optional;

import java.util.Optional;

/**
 * 演示 Optional 的基本操作

 * Author : 动力节点老崔

 */
public class Test01 {
    public static void main(String[] args) {
        //1)把一个字符串封装为 Optional 对象

        Optional<String> ofString = Optional.of("wkcto");     //参数不能为 null


        //2)为指定的值创建 Optional 对象,如果参数为 null,返回空的 Optional 对象

        Optional<String> ofString2 = Optional.ofNullable(null);        //参数可以为 null

        System.out.println(ofString2);          //Optional.empty


        //3)直接创建一个空的 Optional 对象

        Optional<String>    ofString3 = Optional.empty();
        System.out.println( ofString3 );


        //4)get()  获得 Optional 对象中的值,如果值不存在会产生异常

        String    text = ofString.get();
        System.out.println(text);           //wkcto
//          text = ofString2.get();             //java.util.NoSuchElementException


        //5)orElse(),如果 Optional 对象中有值就返回,没有则返回指定的其他值

        text = ofString.orElse("another");
        System.out.println( text );             //wkcto
        text = ofString2.orElse("another");
        System.out.println( text );         //another
```

```
        //6)orElseGet(),如果有值就返回,如果 Optional 对象中没值则创建一个新的
        text = ofString2.orElseGet(() -> "newString");
        System.out.println( text );          //newString

        //7)orElseThrow(),如果值存在就返回,否则抛出异常
//          text = ofString2.orElseThrow(NullPointerException::new);
        text = ofString.orElseThrow(NullPointerException::new);
        System.out.println( text );

        //8)filter(),如果 Optional 对象有值返回满足指定条件的 Optional 对象，否则返回空
的 Optional 对象
        text = ofString.filter(s -> s.length() > 10).orElse("lenth is letter than 10");
        System.out.println( text );
        text = ofString.filter(s -> s.length() > 3).orElse("lenth is letter than 3");
        System.out.println( text );

        //9) map()  如果 Optional 对象的值存在,执行 mapper 映射函数
        text = ofString.map(x -> x.toUpperCase()).orElse("Failure");
        System.out.println( text );                //WKCTO

        text = ofString2.map(x -> x.toUpperCase()).orElse("Failure");
        System.out.println( text );                //Failure

        //10)ifPresent()  如果 Optional 对象有值就执行 Consumer 函数
        ofString.ifPresent(s -> System.out.println("处理数据" + s));
        ofString2.ifPresent(s -> System.out.println("处理数据" + s));          //没有值什么也
不做
        System.out.println("optional");
    }
}
```

Optional 示例 1

```
package com.wkcto.optional;

import javax.print.attribute.standard.NumberUp;
import javax.sound.midi.Soundbank;
import java.util.Optional;

/**

 * Optional 练习 1

 * Author : 动力节点老崔

 */
public class Test02 {
    public static void main(String[] args) {

        Address address = new Address("Beijing", "大族企业湾");
//          User user = new User("laocui", address);
//          User user = new User();
        User user = null;

        System.out.println( getName1(user));
        System.out.println( getName2(user));
        System.out.println( getCity1(user));
        System.out.println( getCity2(user));

    }

    //定义方法返回指定用户的用户名 ,如果用户名不存在返回 unknown

    public   static   String getName1( User user){

        if ( user == null ){            //判断参数接收的 User 对象是否为 null
            return "unknown";
        }
        return   user.name;
    }

    //Optional 可以解决空指针问题

    public   static   String getName2(User user){

        return Optional.ofNullable(user)        //把参数接收的 user 对象包装为 Optional 对象
```

```
                .map(u -> u.name)                    //映射,只需要用户名
                .orElse("unknown");                  //存在就返回,不存在返回 unknown
    }
    //定义方法返回指定用户的城市
    public static    String getCity1(User user){
        if ( user != null){
            if (user.address != null){
                return user.address.city;
            }
        }
        return "unkown";
    }
    //使用 Optional 返回用户的城市
    public static    String getCity2(User user){
        return Optional.ofNullable(user)          //把参数对象包装为 Optional 对象
                .map(u -> u.address)                 //映射用户的地址
                .map(addr -> addr.city)              //映射地址的城市
                .orElse("Unkown");                   //有就返回,没有返回 Unknown
    }
}
```

```
//定义用户类
class    User{
    String name;
    Address address;

    public User(String name, Address address) {
        this.name = name;
        this.address = address;
    }

    public User() {
    }
}
```

```
//定义地址类
class Address{
    String city;
    String house;

    public Address(String city, String house) {
        this.city = city;
        this.house = house;
    }
}
```

Optional 在实际开发中的应用

```
package com.wkcto.optionalapp2;

import java.util.Optional;

/**
 * 每个人可能有一部车,每辆车可能都有保险,每个保险公司都有自己的名称

 * Author : 动力节点老崔

 */
public class Test {
    public static void main(String[] args) {

        //1)创建保险对象

        Insurance in1 = new Insurance();
        in1.setName("pingan");

        Optional<Insurance> insurance1 = Optional.ofNullable(in1);    //有保险

        Optional<Insurance> insurance2 = Optional.ofNullable(null);     //保险为 null


        //2)创建小汽车对象

        Car car1 = new Car("Geely", insurance1);              //Geely 汽车有保险

        Car car2 = new Car("Haval", insurance2);              //Haval 汽车没有保险
```

```
        Optional<Car> carOptional1 = Optional.ofNullable(car1);        //车有保险

        Optional<Car> carOptional2 = Optional.ofNullable(car2);        //车无保险

        Optional<Car> carOptional3 = Optional.ofNullable(null) ;       //没有车


        //3)创建 Person 对象

        Person p1 = new Person("lisi", carOptional1);            //lisi 有车,有保险

        Person p2 = new Person("wangwu", carOptional2);          //wangwu,有车,没有保险

        Person p3 = new Person("zhaoliu", carOptional3);         //zhaoliu,  没有车


        //4)使用 Optional 包装 Person 对象

        Optional<Person> person1 = Optional.ofNullable(p1);        //有车,有保险

        Optional<Person> person2 = Optional.ofNullable(p2);        //有车,没有保险

        Optional<Person> person3 = Optional.ofNullable(p3);        //没有车


        //5)获得人的汽车品牌

        System.out.println(     person1.map(Person::getCar));              //  返  回  一  个
Optional<Optional<Car>>类型的数据

        System.out.println( person1.flatMap(Person::getCar));    //Optional<Car>

System.out.println( person1.flatMap(Person::getCar).map(Car::getBrand).orElse("Unknown"));

System.out.println( person2.flatMap(Person::getCar).map(Car::getBrand).orElse("Unknown"));

System.out.println( person3.flatMap(Person::getCar).map(Car::getBrand).orElse("Unknown"));


        //6)获得人的汽车的保险的名称

        System.out.println( person1.flatMap(Person::getCar)
```

```
                .flatMap(Car::getInsurance)
                .map(Insurance::getName)
                .orElse("Unknwon"));
        System.out.println( person2.flatMap(Person::getCar)
                .flatMap(Car::getInsurance)
                .map(Insurance::getName)
                .orElse("Unknwon"));
        System.out.println( person3.flatMap(Person::getCar)
                .flatMap(Car::getInsurance)
                .map(Insurance::getName)
                .orElse("Unknwon"));

    }
}
```

```
package com.wkcto.optionalapp2;

import java.util.Optional;

/**
 * 定义人类

 * Author : 动力节点老崔

 */
public class Person {
    private String name;

    private Optional<Car> car;          //不确定是否有车


    public Person(String name, Optional<Car> car) {
        this.name = name;
        this.car = car;
    }

    public String getName() {
        return name;
    }

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

```
    }

    public Optional<Car> getCar() {
        return car;
    }

    public Person setCar(Optional<Car> car) {
        this.car = car;
        return this;
    }
}
```

```
package com.wkcto.optionalapp2;

import java.util.Optional;

/**
 * 汽车类
 * Author : 动力节点老崔
 */
public class Car {

    private String brand;           //汽车品牌

    private Optional<Insurance>    insurance;          //不能确定每辆汽车都有保险


    public Car(String brand, Optional<Insurance> insurance) {
        this.brand = brand;
        this.insurance = insurance;
    }

    public String getBrand() {
        return brand;
    }

    public Car setBrand(String brand) {
        this.brand = brand;
        return this;
    }
```

```
    public Optional<Insurance> getInsurance() {
        return insurance;
    }

    public Car setInsurance(Optional<Insurance> insurance) {
        this.insurance = insurance;
        return this;
    }
}
```

```
package com.wkcto.optionalapp2;

/**
 * 保险类

 * Author : 动力节点老崔

 */
public class Insurance {
    private    String name;

    public String getName() {
        return name;
    }

    public Insurance setName(String name) {
        this.name = name;
        return this;
    }
}
```

2.6 对方法参数的反射

在 JDK8 中增加了 Parameter 参数类

```
package com.wkcto.parameter;
```

```
import java.lang.reflect.Method;
import java.lang.reflect.Modifier;
import java.lang.reflect.Parameter;

/**
 * 反射方法中参数
 *  需要在编译时使用-parameters 参数
 * Author : 动力节点老崔
 */
public class Test {
    public static void main(String[] args) {
        //1)创建 Class 对象
        Class<?> claxx = MyClass.class;

        //2)反射所有的方法
        Method[] declaredMethods = claxx.getDeclaredMethods();
        for( Method   method : declaredMethods ){
            //方法的修饰符
            int mod = method.getModifiers();
            System.out.print(Modifier.toString(mod) + " ");
            //方法返回值类型
            Class<?> returnType = method.getReturnType();
            System.out.print( returnType.getSimpleName() + " ");
            //方法名
            System.out.print( method.getName());

            //方法参数
            System.out.print("(");
            Parameter[] parameters = method.getParameters();
            for(int i = 0 ; i < parameters.length; i++){
                System.out.print( parameters[i].getType().getSimpleName()
+ " ");
                System.out.print( parameters[i].getName());
                //参数之间使用逗号分隔
                if ( i < parameters.length - 1 ){
                    System.out.print(",");
                }
            }

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

```
    }
}
```