

day12_Lambda_王世杰

T1.Lambda表达式相关的简答题：

1. Lambda表达式语法创建的是什么？是一个类吗？是对象吗？

对象

2. 对于语法 `() -> {}`，描述一下 `()` 和 `{}` 的简化。

1. `()` 是抽象方法的形参列表：

(1) 只有一个形参：`a -> {}`

(2) 两个以上形参省略数据类型：`(a,b) -> {}`

2. `{}` 是方法体：

(1) 只有一条语句直接 `() ->` 特殊的，当这一条语句就是方法的返回值语句时，那么 `{}` 和 `return` 一起省略。

(2) 多条语句可以封装成方法，直接引用：

(形参列表) `->` 已实现的方法(形参列表)；

方法的归属者：`::`方法名；

3. 描述一下Lambda表达式的方法引用。尤其说明一下什么样的方法可以作为Lambda表达式的实现指向。

Java允许Lambda表达式的抽象方法的实现可以直接指向一个已经存在的方法，而不是自己书写实现。这种语法在Java中称之为“**方法引用**”！

1. 访问权限修饰符和`static`之类的修饰符，实际没有多大影响，但是肯定需要访问权限。

2. 返回值类型，应该保持一致。如果抽象方法返回一个父类引用类型，那么这个已实现的方法可以返回子类类型。

3. 方法名是什么不重要。

4. 形参列表，必须保持一致，数据类型，位置必须严格对应，但是形参名无所谓。

5. 方法体，无所谓，自己重写即可。

T2.Lambda表达式的练习

提供以下6个功能接口，请用Lambda表达式分别创建它们的子类对象，调用`test()`方法进行测试。

//无返回值无参数的功能接口

```

@FunctionalInterface
interface INoReturnNoParam {
    void test();
}

//无返回值有一个参数的功能接口
@FunctionalInterface
interface INoReturnOneParam {
    void test(int a);
}

//无返回值两个参数的功能接口
@FunctionalInterface
interface INoReturnTwoParam {
    void test(int a, int b);
}

//有返回值无参数的功能接口
@FunctionalInterface
interface IHasReturnNoParam {
    int test();
}

//有返回值一个参数的功能接口
@FunctionalInterface
interface IHasReturnOneParam {
    int method(int a);
}

//有返回值两个参数的功能接口
@FunctionalInterface
interface IHasReturnTwoParam {
    int test(int a, int b);
}

```

注意：自由发挥方法实现，主要练习Lambda表达式的语法。

```

package com.cskaoyan.homework.day12;

public class LambdaTest {

    public static void main(String[] args) {
        //无返回值无参数的功能接口
        INoReturnNoParam iIf1 = () -> System.out.println("Lambda表达式
无返回值无参数的功能接口.");
        iIf1.test();

        //无返回值有一个参数的功能接口

```

```

        INoReturnOneParam iIf2 = a -> System.out.println("Lambda表达式
无返回值有一个参数的功能接口: a = " + a);
        iIf2.test(10);

        //无返回值两个参数的功能接口
        INoReturnTwoParam iIf3 = (a, b) -> System.out.println("Lambda
表达式无返回值两个参数的功能接口:" + a + " + " + b + " = " + (a + b));
        iIf3.test(10, 20);

        //有返回值无参数的功能接口
        IHasReturnNoParam iIf4 = () -> 10;
        System.out.println("Lambda表达式有返回值无参数的功能接口: 返回值为" + iIf4.test());

        //有返回值一个参数的功能接口
        IHasReturnOneParam iIf5 = a -> a;
        System.out.println("Lambda表达式有返回值一个参数的功能接口: 返回值为" + iIf5.method(10));

        //有返回值两个参数的功能接口
        IHasReturnTwoParam iIf6 = (a, b) -> (a + b);
        System.out.println("Lambda表达式有返回值两个参数的功能接口: 返回值为" + iIf6.test(10, 10));
    }
}

//无返回值无参数的功能接口
@FunctionalInterface
interface INoReturnNoParam {
    void test();
}

//无返回值有一个参数的功能接口
@FunctionalInterface
interface INoReturnOneParam {
    void test(int a);
}

//无返回值两个参数的功能接口
@FunctionalInterface
interface INoReturnTwoParam {
    void test(int a, int b);
}

//有返回值无参数的功能接口
@FunctionalInterface
interface IHasReturnNoParam {
    int test();
}

```

```
//有返回值一个参数的功能接口
@FunctionalInterface
interface IHasReturnOneParam {
    int method(int a);
}

//有返回值两个参数的功能接口
@FunctionalInterface
interface IHasReturnTwoParam {
    int test(int a, int b);
}
```

```
Run: LambdaTest x
C:\Users\h8916\.jdk\semeru-1.8.0_322\bin\java.exe ...
Lambda表达式无返回值无参数的功能接口。
Lambda表达式无返回值有一个参数的功能接口：a = 10
Lambda表达式无返回值两个参数的功能接口：10 + 20 = 30
Lambda表达式有返回值无参数的功能接口：返回值为10
Lambda表达式有返回值一个参数的功能接口：返回值为10
Lambda表达式有返回值两个参数的功能接口：返回值为20
```

T3.这道题目实际课堂上已经讲解过了，你能自己实现一下它吗？

- 1 定义一个计算（compute）接口，接口中有加减乘除四个抽象方法。
- 2 然后使用匿名内部类去实现加减乘除并测试

上述功能，昨天已经实现了，今天考虑用lambda改进它，如下：

- 用以下功能接口，实现一个计算器的工具类
- 在工具类中，只需要一个工具方法，就能够实现所有的计算功能

```
1 @FunctionalInterface
2 interface Compute {
3     double compute(double a, double b);
4 }
```

工具类和工具方法如下：

```
1 //需要提供一个使用功能接口的方法完成需求
2 class ComputeTool {
3     private ComputeTool() {
4     }
5
6     public static void calc(Compute com, double a, double b) {
7         //...
8     }
9 }
```

你知道如何使用吗？联系一下，今天讲解的String数组的过滤，**体会Lambda表达式对象表示的规则。**

```
package com.cskaoyan.homework.day12;

public class LambdaCalTest {

    public static void main(String[] args) {
        Computer com1 = LambdaCalTest::add;
        ComputerTool.calc(com1, 10, 10);

        Computer com2 = LambdaCalTest::subtract;
        ComputerTool.calc(com2, 10, 10);

        Computer com3 = LambdaCalTest::multiply;
        ComputerTool.calc(com3, 10, 10);

        Computer com4 = LambdaCalTest::divide;
        ComputerTool.calc(com4, 10, 10);

    }

    private static double divide(double a, double b) {
        return a / b;
    }

    private static double multiply(double a, double b) {
        return a * b;
    }

    private static double subtract(double a, double b) {
        return a - b;
    }

    private static double add(double a, double b) {
        return a + b;
    }
}

@FunctionalInterface
interface Computer {
    double compute(double a, double b);
}

class ComputerTool {
    private ComputerTool() {

    }

    public static void calc(Computer com, double a, double b) {
        System.out.println(com.compute(a, b));
    }
}
```

```
}
```

```
Run: LambdaCalTest x
C:\Users\h8916\.jdk\semeru-1.8.0_322\bin\java.exe ...
20.0
0.0
100.0
1.0
```

T4.扩展:完成以下练习

参考课堂上讲的，使用Lambda表达式实现过滤数组元素的方案，实现数组的映射。

将一个对象数组，映射成另一个对象数组。

比如：

将一个Student对象数组，映射为装所有学生的成绩的数组。

将一个String对象数组，映射为装所有字符串对象长度的数组。

.....

所以这个映射的规则就是将一个Object对象转换为另一个Object对象，将这个规则应用到整个数组中，就完成了数组元素的映射。

当然，这对于大家而言，不算简单，思考思考，不行可以直接参考一下我的实现。

```
package com.cskaoyan.homework.day12;

import java.util.Arrays;

public class MapTest {

    public static void main(String[] args) {

        //映射 学生数组 -> 分数数组
        Student[] stus1 = new Student[5];
        stus1[0] = new Student("张三",18,95.5);
        stus1[1] = new Student("李四",19,93);
        stus1[2] = new Student("王五",17,94.5);
        stus1[3] = new Student("赵六",19,96.5);
        stus1[4] = new Student("黄七",18,97);
        Double[] scores = MapTest.mapObjectArr(stus1,o -> {
            Student stu = (Student)o;
            return stu.getScore();
        });
    }
}
```

```

    });
    printStudents(stus1);
    printDoubleArr(scores);

    //将一个String对象数组，映射为装所有字符串对象长度的数组。
    String[] strs = {"string", "int", "byte", "float", ""};
    Double[] strsLength = mapObjectArr(strs, o -> {
        String str = (String) o;
        return (double) str.length();
    });
    System.out.println(Arrays.toString(strs));
    printDoubleArr(strsLength);
}

private static void printStudents(Student[] stus1) {
    System.out.println("打印Student[]数组:");
    for (Student student : stus1) {
        System.out.println("姓名:"+student.getName()+"，年
龄:"+student.getAge()+"，分数:"+student.getScore());
    }
}

public static Double[] mapObjectArr(Object[] o, Map map){
    int count = o.length;
    Double[] scores = new Double[count];
    for (int i = 0; i < o.length; i++) {
        scores[i] = map.map(o[i]);
    }
    return scores;
}

public static void printDoubleArr(Double[] scores){
    System.out.println("打印Double[]数组:");
    for (Double score : scores) {
        System.out.println(score);
    }
}

}

interface Map{
    Double map(Object o);
}

class Student{
    String name;
    int age;
    double score;
}

```

```
public Student(String name, int age, double score) {  
    this.name = name;  
    this.age = age;  
    this.score = score;  
}  
  
public Double getScore() {  
    return score;  
}  
  
public String getName() {  
    return name;  
}  
  
public int getAge(){  
    return age;  
}  
}
```



```
Run: MapTest x
C:\Users\h8916\.jdk\semeru-1.8.0_322\bin\java.exe ...
打印Student[]数组:
姓名:张三, 年龄:18, 分数:95.5
姓名:李四, 年龄:19, 分数:93.0
姓名:王五, 年龄:17, 分数:94.5
姓名:赵六, 年龄:19, 分数:96.5
姓名:黄七, 年龄:18, 分数:97.0
打印Double[]数组:
95.5
93.0
94.5
96.5
97.0
[string, int, byte, float, ]
打印Double[]数组:
6.0
3.0
4.0
5.0
0.0
```

T5.以下简答题,请回答:

1. 描述一下运行时Class对象。

在 Java 中要使用一个类首先要将该类加载到内存中, 系统会为该类生成一个 `java.lang.Class` 的实例. 这个 `Class` 对象的作用很大, 通过它系统可以访问到 JVM 中该类的信息, 同时 `Class` 对象也是实现 Java 反射机制的核心要素。

2. 某个类Class对象为什么具有唯一性?

只会在类加载的时候生成。

3. Class对象的唯一性可以用来做什么?

判断两个对象是否是一个类。

4. equals方法的排它性有两种实现方法，请分别描述。

```
if(c1.getClass() != obj.getClass)    return false;
if (!(obj instanceof ClassName))    return false;
```

T6.getClass()方法练习

定义两个类，然后分别创建对象，调用getClass方法

用“==”号比较它们的运行时Class对象是否相等，并说明原因

理解运行时类对象、类加载、类的对象的区别

```
package com.cskaoyan.homework.day12;

public class EqualsTest {
    public static void main(String[] args) {
        Cat c1 = new Cat("Tom", 2);
        Cat c2 = new Cat("Tom", 2);
        System.out.println("c1 == c2 结果:");
        System.out.println(c1 == c2);
        System.out.println("c1.getClass() == c2.getClass() 结果:");
        System.out.println(c1.getClass() == c2.getClass());
    }
}

class Cat {
    String name;
    int age;

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

c1 == c2 :不相等 因为默认调用toString()方法,结果不一样

c1.getClass() == c2.getClass() :相等 因为Cat类只有一个Class类对象

T7.equals方法练习

定义一个Animal类

成员变量:

int age,String name,double price

手写它的equals方法, 比较getClass和instanceof的区别

```
package com.cskaoyan.homework.day12;

public class Equals {
    public static void main(String[] args) {
        Animal a1 = new Animal(2,"Tom",1000);
        Dog a2 = new Dog(2,"Tom",1000);
        Animal a3 = new Animal(1,"Tom",1000);
        Animal a4 = new Animal(2,"Tom",2000);
        Animal a5 = new Animal(2,"Jack",1000);
        Animal a6 = new Animal(2,"Tom",1000);
        Animal1 a7 = new Animal1(2,"八戒",1000);
        Pig a8 = new Pig(2,"八戒",1000);
        // a1 = a1
        System.out.print("a1 == a1 :");
        System.out.println(a1.equals(a1)+"\n");
        // age
        System.out.print("判断age:");
        System.out.println(a1.equals(a3)+"\n");
        //price
        System.out.print("判断price:");
        System.out.println(a1.equals(a4)+"\n");
        //name
        System.out.print("判断name:");
        System.out.println(a1.equals(a5)+"\n");
        // 全部通过
        System.out.print("判断全部通过:");
        System.out.println(a1.equals(a6)+"\n");
        // instanceof
        System.out.print("用instanceof判断类相同或者子类:");
        System.out.print(a1.equals(a2)+"\n");
        // getClass()
        System.out.print("用getClass()判断类相同");
        System.out.println(a7.equals(a8));
    }
}

class Animal {
    int age;
    String name;
    double price;

    public Animal(int age, String name, double price) {
```

```

        this.age = age;
        this.name = name;
        this.price = price;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }
        if (!(o instanceof Animal)) {
            return false;
        }
        Animal animal = (Animal) o;
        if (age != animal.age) {
            return false;
        }
        if (Double.compare(animal.price, price) != 0) {
            return false;
        }
        return name.equals(animal.name);
    }
}

class Dog extends Animal {

    public Dog(int age, String name, double price) {
        super(age, name, price);
    }

}

class Animal1 {
    int age;
    String name;
    double price;

    public Animal1(int age, String name, double price) {
        this.age = age;
        this.name = name;
        this.price = price;
    }

    @Override
    public boolean equals(Object o) {
        if (this == o) {
            return true;
        }

```

```



        if (o.getClass() != this.getClass()) {
            return false;
        }
        Animal animal = (Animal) o;
        if (age != animal.age) {
            return false;
        }
        if (Double.compare(animal.price, price) != 0) {
            return false;
        }
        return name.equals(animal.name);
    }
}



class Pig extends Animal1 {



    public Pig(int age, String name, double price) {
        super(age, name, price);
    }
}
}



```



Run: Equals x





C:\Users\h8916\.jdk\semeru-1.8.0_322\bin\java.exe ...




a1 == a1 :true







判断age:false



判断price:false




判断name:false

判断全部通过:true

用instanceof判断类相同或者子类:true

用getClass()判断类相同false

