Lambda 表达式

Lambda 表达式介绍

Java 函数式编程的核心概念之一，Lambda 表达式的出现是为了简化函数式接口的。

Interface 接口的基本概念

接口是用来定义一个协议或者约定，它只声明方法但不提供方法的具体实现，我们称这种方法为抽象方法（Abstract Method）。具体方法的实现是由实现这个接口的 Class 来提供。

这样可以讲接口的声明和具体实现分开，业务中就只需要关心方法，而不需要关心具体的实现类，从而实现代码的解耦和模块化。

```
/**
 * 需求：通过多种方法来发送不同形式的消息，比如既可以发送 Email 邮件消息，又可以发送
SMS 短信
 */

public interface Message{
    void send();
}

public class Email implements Message {
    String email;

    public Email() {

    }

    public void send() {
        System.out.println("This is an email.");
    }
}

public class Sms implements Message {
    String phoneNumber;

    public Sms() {

    }

    public void send() {
        System.out.println("This is a sms.");
    }
}

/**
 * Main 方法：统一发送消息
 */
```

```
public class Main {
    public static void main(String[] args) {
        Message email = new Email();
        sendMessage(email);

        Message sms = new Sms();
        sendMessage(sms);

    }

    /**
     * 因为参数类型为 Message 接口，这意味着不管传入的是 Email 还是 SMS 对象
     * 二者的类已经各自实现了 Message 接口的 send 方法，
     * 所以 sendMessage 方法内部只需要执行 message 的 send 方法
     * 而不需要关心执行的到底是哪个对象的 send 方法，免去的复杂的调用和判断
     * 这是面向接口编程的典型应用场景之一
     */
    static void sendMessage(Message message){
        message.send();
    }
}
```

缺陷：上述代码中，按照传统方法已经正确实现了和使用了接口，应用场景也正确。但要发送一条消息每次都需要好几个步骤（以 sms 消息为例）：

- 先创建 Sms 类
- 实现 send 方法
- 实例化 sms
- 发送

为什么要引入 Lambda 表达式

Lambda 表达式的出现可以帮助我们解决上述问题：它可以提供一种快速简洁的方式来实现上述接口，**接口抽象方法的具体实现直接在 Lambda 表达式里定义即可**。而不用像传统方法那样创建类、实现接口的抽象方法、实例化、发送。

```
/**
 * Main 方法：统一发送消息
 */
public class Main {
    public static void main(String[] args) {
        // 此处不在需要实例化 Email 或者 Sms 类
        // Message email = new Email();
        // sendMessage(email);

        // Message sms = new Sms();
        // sendMessage(sms);

        // 可以在 lambda 表达式中直接实现 Message 接口的 send 抽象方法，然后作为参数传递给 sendMessage() 方法。
        // 等效于之前实现的 Email 实例
```

```
        sendMessage(() -> {
            System.out.println("This is an Email.");
        });

    }

    /**
     * 因为参数类型为 Message 接口，这意味着不管传入的是 Email 还是 SMS 对象
     * 二者的类已经各自实现了 Message 接口的 send 方法，
     * 所以 sendMessage 方法内部只需要执行 message 的 send 方法
     * 而不需要关心执行的到底是哪个对象的 send 方法，免去的复杂的调用和判断
     * 这是面向接口编程的典型应用场景之一
     */
    static void sendMessage(Message message){
        message.send();
    }
}
```

Lambda 表达式语法

`() -> {函数体}`

此处 `()` 便是 `send()` 处的实现

- 如果函数体中只包含一条语句，则可以省略大括号，否则不可以省略大括号
- 抽象方法中有参数时，对应的 lambda 表达式中也应该带有参数，此时lambda表达式中可以不写参数类型，编译器会自动推断。
- 如果抽象方法中只有一个参数，则可以去掉 `()`
- 抽象方法中包含多个参数时，`()` 不可以省略，`sendMessage((name,title) -> {System.out.println("This is an email to " + title + " " + name);});`
- 抽象方法有返回值时
- Lambda 表达式可以像普通对象那样赋值给变量

```
public class Main {
    public static void main(String[] args) {

        Message lambda = (name,title) -> {
            System.out.println("This is an email to " + title + " " +
name);
        };
        sendMessage(lambda);

    }

    static void sendMessage(Message message){
        message.send("Albert", "Mr");
    }
}
```

函数式接口与 Lambda 表达式

lambda 表达式只能应用于有且只有一个抽象方法的接口上，我们称这样的接口为**函数式接口**

当一个接口有多个抽象方法时，java 将无法确定 lambda 意图实现哪一个方法，当只有一个方法时，意图非常明确。

换言之只有函数式接口才能使用 lambda 表达式。

我们可以在接口顶部使用`@FunctionalInterface`注解，该方法是可选的，函数失接口并不依赖该注解。只要接口定义符合函数式接口的定义便可以使用 lambda 表达式。

但是日常使用中推荐使用 `@FunctionalInterface` 注解，使用该注解，即可以帮助开发人员理解接口设计意图，也可以帮助 debug。因为如果增加注解后，但接口定义不符合函数式接口定义，如果没有抽象方法或者是出现多余的抽象方法将会报错。

Lambda 表达式归根结底是一种语法糖，用于简化函数式接口的实现。Java 标准库中也包含了许多内置的函数式接口，比如 `Predicate, Function, Consumer, Supplier` 等，经常与`Stream API` 配合使用。

常用技术术语

Lambda Expressions

Functional Interface

Abstract Method

() Parentheses

- Hyphen