21 接口和面向接口编程

当我们探讨接口时候，通常有几种：

- 一个库或者模块的对外API接口，通过接口通信，隐藏内部工作细节
- 一些语言的关键字，例如Java的interface来实现类与类的联系
- 面向接口编程中的接口，这里的接口比较抽象，设计模式指的是：接口是对象能够响应的请求的集合

本章主要探讨2和3点，这里从Java语言讲解开始，因为JS没有这一层

回到Java的抽象类

之前1.3小结我们通过Java实现了一个动物世界，还是makeSound的例子，我们只有通过向上转型，编写一个父类Animal类，才能将对象（鸭子，鸡，麻雀）的具体类型隐藏在超类之后，并在类型检查系统下相互替换使用，展现多态性，更灵活。

这里重点是了解Java抽象类的作用：

- 向上转型：让Duck和Chicken对象的类型隐藏在Animal类型身后，这样duck和chickken才能交换使用
- 建立契约：继承抽象类要求必须复写抽象方法，这种契约可以帮助我们提高可靠性，编译器也会提前校验

> 🐵 不关注对象的具体类型，仅仅针对超类中的“契约”来编写程序，减少子系统实现之间的依赖，这便是本章探讨的：面向接口编程，而非面向实现编程

注意这里虽然讨论的是抽象类，但是实际上面向接口编程的接口是一个抽象概念，其实是“面向超类型编程”。也可以看成面向抽象编程，针对契约去做设计但是不关心具体如何去做

Interface

Java中也可以使用interface关键字来实现同样效果，即接口继承。

一个类可以实现多个interface，它本身是抽象概念更进一步，不提供任何具体实现和方法体，但是允许创建者明确方法名，参数列表和返回类型，提供约定而不关心实现

同样也可以用于向上转型，也是表达多态性的一条途径

JS是否需要抽象类与interface

对于JS来说，它是动态类型语言，所以类型本身就是相对模糊的概念。所以说“向上转型”其实不是刚需，根本上向上转型都是Object类型：

```
Object ary = new Array();
Object date = new Date();
```

很少有人在JS开发中关心对象的真正类型，而对象的多态也是与生俱来的在动态语言中。

所以接口在JS中作用就剩下一个检查代码的规范性了，例如对象是否实现了某个方法，函数参数是否与预期相符等等，不然很容易出各种bug

现在我们有了TS配合编译器其实可以静态帮助我们检查代码的规范性了，通过适当规范类型也可以实现静态语言的检查了

用鸭子类型进行接口检查

鸭子类型的概念我们之前已经了解过了：

> 如果它走起来像鸭子，叫起来也是鸭子，那么它就是鸭子了

鸭子类型是动态类型语言中一个概念，不必借助超类帮助来实现：面向接口编程而非面向实现编程

例如，一个对象有push和pop方法并实现正确，那么我们就可以拿它当做栈使用。

当然在JS中总是进行接口检查会很冗余，目前主流方式还是借助TS和Type/Interface来进行接口检查

用TS编写基于interface的命令模式

这里我们拿命令模式作为示例，在JS中我们一般通过闭包+高阶函数来实现这个模式，通过命令模式将子功能交出去给独立的程序员实现，约定子功能都有execute方法来执行命令。

为了防止存在粗心没有实现execute方法的情况，我们需要加上类似如下代码校验：

```
if(typeof command.execute !== 'function'){
    throw new Error('没有实现execute')
}
```

这里我们其实可以通过TS来代替这种防御性代码：

```
interface Command {
    execute:function;
```

```
}
```

然后让子类实现这个interface：

```
class RefreshMenuCommand extends Command {
    execute(){
        //...
    }
}

class CloseMenuCommand extends Command {
    // 忘记写了
}
```

这里TS的编译器会在代码编写就给出相应的错误提示。