第 3 章

CHAPTER 3

语法基础：ArkTS 应用开发语言

时间追溯到 2019 年，HarmonyOS 1.0 正式发布，彼时 HarmonyOS 应用主要采用的编程语言为 JavaScript 语言。直至 2022 年，华为正式推出自主研发的编程语言——ArkTS，其中，Ark 为鸿蒙操作系统的开发代号，TS 则为 TypeScript 的缩写。

ArkTS 是在 TypeScript 的基础上扩展的声明式语言。ArkTS 不仅兼容 JavaScript、TypeScript 语言生态，还扩展了声明式 UI 语法和状态管理等相应能力，从而使跨端界面开发及并行化任务开发更高效简洁。

到了 2024 年，随着 HarmonyOS Next 星河版的发布，华为正式将 ArkTS 作为鸿蒙原生应用开发的主要推荐编程语言。相较于类 Web 开发范式的 JavaScript 语言，采用声明式开发范式的 ArkTS 更加适合开发复杂度较高、团队合作度较高的应用程序。

本章将主要介绍 ArkTS 编程语言的相关知识，旨在帮助读者快速入门 ArkTS 语言。编程语言的学习可能相较于应用开发过程略显枯燥，但良好的语法基础将对后续的开发过程起到事半功倍的效果。

本章节将创建一个名为 LearnArkTS 的工程项目，并以此项目为基础进行讲解和分享。

3.1 参数声明

在开发过程中，当需要在多个地方使用同一个参数值时，通常的做法是将其作为单独的参数进行声明。在 ArkTS 中，声明参数的关键字有两种：let 和 const。

let 关键字用于声明一个可变的参数，该参数声明会被赋予一个初始值，并在使用过程中可以被重新赋值，从而显示一个新的值。

const 关键字用于声明一个不可变的参数，使用 const 关键字声明的参数，其初始被赋予的值在之后的使用过程中不允许被更改。

若对 Java 编程语言有所了解，则会发现 let 和 const 对应着 Java 中的非 final 变量和 final 变量，而 ArkTS 与 Java 不同之处在于，声明式语法的特点是声明参数时需要明确参数的类型。

演示参数声明方式，代码如下：

```
let userName: string = "Ricardo";
const pi: number = 3.14159;
```

在上述代码中，声明了一个 string 类型的可变参数 userName，并赋予了初始值 "Ricardo"；同时声明了一个 number 类型的不可变参数 pi，其初始值为 3.14159。

在 ArkTS 中声明参数时，由于其具有优秀的类型推导机制，它可以根据参数值的内容自动判断参数的类型，代码如下：

```
let userName = "Ricardo";
const pi = 3.14159;
```

在上述代码中，并未告知 ArkTSuserName 和 pi 参数的类型，ArkTS 根据 userName 参数的值是一个 string 类型，自动判断 userName 参数应为 string 类型，同理 pi 参数也是如此。

在 ArkTS 中，常见的参数类型有 string(字符串)、number(数值)、boolean(布尔类型)、array(数组)等，在后续章节中会经常使用。

3.2 函数定义

函数是代码执行的载体，当需要实现具有一定逻辑的功能时会将逻辑代码写在一个函数中，该函数将作为逻辑方法被调用。以可变参数为例，可以先声明一个可变的参数，并定义一个函数实现可变参数值的重新赋值，代码如下：

```
let isTap: boolean = false;
function toggleIsTap() {
    isTap = !isTap;
}
```

在上述代码中，由于 isTap 参数使用 let 关键字进行声明，因此在代码逻辑中允许开发者在使用时给该参数重新赋值。故可以定义一个函数 toggleIsTap，在其函数体中给 isTap 重新赋值为!isTap(非运算，即取相反的值，原 isTap 的值为 false，非运算后 isTap 的值为 true)。

在此讲解函数定义的语法规则。定义函数使用的关键字为 function，这是定义函数必须遵循的规则。在 function 关键字后为函数名称，函数名称建议使用“驼峰命名法”，即以英文小写开头，随后英文首字母大写。函数名称需具有一定的辨识度，与实现的功能逻辑紧密相关，这是良好的编码习惯。

函数名称后面的圆括号内通常作为接收参数的内容，即该函数由于代码逻辑或实现功能需要传入的参数，此部分为可选。如上述代码所示，代表着该函数无须接收任何参数。

紧接着的花括号及花括号中的内容称为函数体，函数的具体业务逻辑都会在函数体中实现，例如在上述代码中实现了给 isTap 参数重新赋值的逻辑。

再举一个简单的例子，计算两个数的和，代码如下：

```
let sum: number = 0;
function addNumbers(num1: number, num2: number) {
    sum = num1 + num2;
}
```

在上述代码中，使用 let 关键字声明了一个 number 类型的参数 sum，并赋予了初始值 0。同时定义了一个函数 addNumbers，接收两个 number 类型的参数，在 addNumbers 函数体中实现计算逻辑，并将两个数之和重新赋值给参数 sum。

还可以给函数定义返回值的类型，使其返回一个指定类型的数据，代码如下：

```
function addNumbers(num1: number, num2: number): number {
    return num1 + num2;
}
```

在上述代码中，定义了该函数返回一个 number 类型的数据，并在函数体中使用 return 关键字来实现返回逻辑。这是 ArkTS 中最基本也是最常用的函数用法，在实际开发过程中会经常用到自定义函数，以实现复杂的业务逻辑。

除此之外，ArkTS 还提供了一种灵活的函数定义方式——箭头函数，可以实现隐藏 return 的操作，使代码更加简洁。以 addNumbers 函数为例，可以写成如下形式：

```
let addNumbers = (num1: number, num2: number): number => num1 + num2;
```

在上述代码中，箭头函数使用=>符号来定义，用于指向函数对象本身。箭头函数不会创建自己的 this 绑定，而是继承外部作用域的 this 值。这在处理回调函数和对象方法时可以避免 this 指向错误的问题。箭头函数有助于提高代码的可读性和可维护性，是 ArkTS 的语法糖之一。

3.3　条件判断语句

在参数的基础类型中提到了 boolean 类型，这是一种正确与错误之间条件判断的概念，而在程序执行时，也经常需要使用条件判断的逻辑来让程序做出不同的反馈，例如“判断水果新鲜就吃掉，不新鲜就丢掉”。

ArkTS 的条件判断语句主要有两种：if 和 switch。

3.3.1　if 条件语句

if 条件语句和绝大多数编程语言的 if 语句类似，在 if 条件中执行一条逻辑代码，在 else 条件中执行另一条逻辑代码，在多重条件下，还会用到 else if 进行多重逻辑判断。

举一个简单的例子，假设在一场考试中 60 分为合格线，超过 90 分为优秀，则使用 if 条件语句的代码如下：

```
//第 3 章/Index.ets
function checkScore(score: number) {
      if (score < 60) {
        console.log("不合格");
      } else if (score >= 60 && score < 90) {
        console.log("合格");
      } else if (score >= 90) {
        console.log("优秀");
      } else {
        console.log("无效的成绩");
      }
}
```

在上述代码中,定义了一个函数 checkScore,该函数传入一个 number 类型的参数 score,在函数体中使用 if 语句对 score 参数值的范围进行条件判断。当 score 参数值小于 60 时,则调用 console.log()函数在控制台输出文字内容“不合格”;当 score 参数值大于或等于 60 且小于 90 时,则输出文字内容“合格”;当 score 参数值大于或等于 90 时,则输出文字内容“优秀”;当以上情况都不满足时,则控制台输出文字内容“无效的成绩”。

调用 checkScore()函数的方法也很简单,代码如下:

```
let score = 85;
checkScore(score);
```

在上述代码中,声明了参数 score,调用 checkScore()函数并将 score 作为参数传入函数体中,在实际开发中可以在终端中看到文字内容“合格”。

3.3.2 switch 条件语句

接下来讲解 switch 条件语句。ArkTS 的 switch 条件语句和 Java 的 switch 条件语句类似,用于基于预期的不同结果来执行不同的代码块。

与 if 条件语句不同的是,switch 条件语句是基于预期的结果值进行匹配的,常用于单个表达式的场景,而 if 条件语句则是基于布尔表达式的,常用于处理复杂的逻辑判断场景。

仍以成绩查询为例,将成绩以等级划分为 A、B、C、D,每个等级对应着成绩的层级,则使用 switch 条件语句的代码如下:

```
//第 3 章/Index.ets
function checkGrade(grade: string) {
      switch (grade) {
      case "A":
      console.log("优秀");
      break;
      case "B":
      console.log("良好");
      break;
      case "C":
```

```
        console.log("合格");
        break;
        case "D":
        console.log("不合格");
        break;
        default:
        console.log("无效的成绩");
        }
    }
```

在上述代码中，定义了一个函数 checkGrade，该函数接受一个 string 类型的参数 grade，并在函数体中使用 switch 语句对传入的 grade 参数的值与各个 case 进行匹配。

当 grade 为 A 时，调用 console.log 函数在控制台输出文字内容“优秀”；当 grade 为 B 时，输出“良好”；若 grade 为 C，输出“合格”；当 grade 为 D 时，输出“不合格”；当传入的 grade 的值不匹配上述任何一种情况时，通过 default 分支输出文字内容“无效的成绩”。

调用 checkGrade 函数，只需传入对应的成绩等级的值，就可以在控制台输出相应的文字内容，代码如下：

```
//第 3 章/Index.ets
checkGrade("A");                    //输出:优秀
checkGrade("B");                    //输出:良好
checkGrade("C");                    //输出:合格
checkGrade("D");                    //输出:不合格
checkGrade("F");                    //输出:无效的成绩
```

3.4　循环语句

学习完条件判断语句之后，接下来开始学习 ArkTS 中的循环语句。

提及循环语句，可能首先想到的是 for 循环语句。无论是 Kotlin、Swift 等移动端语言，还是使用最广泛的 Java 语言，for 循环语句无论是在语法上还是在使用方法上几乎没什么差别。基于 TypeScript 的 ArkTS 也是如此。

3.4.1　for 循环语句

for 循环语句用于重复执行某段代码块的场景，直至满足指定条件才结束循环，例如首先声明一个数组来表示成绩，然后使用 for 循环语句来打印数组中的所有成绩，代码如下：

```
const scores = [85, 92, 78, 64, 89];
for (let score of scores) {
    console.log("成绩:", score);
}
```

在上述代码中，定义了一个数组 scores，并为其赋值 5 个成绩。使用 for-of 语句来遍历

数组中的元素，它会找到数组中的所有元素，并赋值给指定的参数。声明了一个参数 score 来接收 scores 数组中的数据，并在控制台中输出 score 的值。

在控制台中可以看到，从第 1 个成绩 85 开始，控制台依次打印输出 92、78、64、89。

注意到，在 for 循环语句的使用中，使用了 for-of 语句来遍历数组中的元素，这非常有效。for-of 语句适合用于 Arrays(数组)、Strings(字符串)、Maps(映射)、Sets(集合)等可遍历的元素，在每次遍历时会返回一个对象的值。

3.4.2 while 循环语句

接下来学习 while 循环语句。与 for 循环语句的使用场景不同，while 循环语句更适合处理条件判断不确定的场景。while 循环语句只在条件满足的情况下才重复执行某段代码块，直至不满足条件时终止循环。

以一个数学计算案例为例，计算整数 1 加到 100 的值，可以使用 while 循环语句循环计算在 1 到 100 以内所有整数的总和，代码如下：

```
//第 3 章/Index.ets
let sum = 0;let number = 1;
while (number <= 100) {
    sum += number;
    number++;
}
console.log("1 到 100 的总和是:", sum);
```

在上述代码中，声明了两个参数 sum、number，分别代表合计数和计算数，两个参数分别被赋予了初始值 0 和 1。

使用 while 循环语句，首先进行条件判断，当满足 number <= 100 的条件时，重复执行内部的代码块。在 while 循环里，每执行一次，当前 number 的值都会被增加到 sum 中，然后 number 的值增加 1。如此，循环会不断地将数字从 1 加到 100。

当 number 的值被增加到 101 时，while 循环将不满足 number <= 100 的条件，循环自动结束。最后可以在控制台输出 sum 的值，即从 1 到 100 的总和，输出结果为 5050。

3.5 面向对象编程

从面向过程编程，到面向对象编程，以及时下 AI 时代下延伸出的“面向用户”编程，编程语言和编程方式都在不断地“进化”，进化到每个普通人，甚至是不懂编程的人都可以借助工具快速开发一款属于自己的软件。尽管现在的 AI 辅助编程已经开始变得普适化了，但对于开发者来讲，掌握基本的编程思维，特别是面向对象编程的思维，依然是必不可少的。

面向对象编程，简单来讲，也就是可以将任何事物都抽象为一个对象，可以给这个对象添加很多属性来描述这个对象是什么，也可以添加一个函数方法来让这个对象实现什么逻

辑，而描述这个对象，可以称为创建一个类。类是对象的一种封装，通过类的封装，既可以对对象的参数进行说明，也可以创建函数来实现某些特定的需求。

3.5.1　类和对象

类的创建比较简单，使用 class 关键字来对类进行声明，代码如下：

```
class Animal {}
```

上述代码是一个空类的实现，声明了一个名为 Animal 的类，此时 Animal 类并没有任何内容。接下来给这个类添加属性来进行描述，例如名字，并且创建一个函数来输出内容，代码如下：

```
//第 3 章/Index.ets
class Animal {
      name = "";
      eat() {
        console.log(`${this.name}正在吃东西`);
      }
}
```

在上述代码中，为 Animal 类创建了 name 属性。虽然未给属性指定类型，但在后续的使用过程中会重新给属性赋值，因此这里的属性的声明默认认为是使用 let 关键字进行声明。

声明相关属性后，还创建了一个函数 eat()，在 eat()函数中，通过模板字符串 ${this.name}来引用 Animal 实例的 name 属性，并输出 name 对应的动物正在吃东西的文字内容。这个函数会根据实例的 name 属性，动态输出相应的内容，例如“某某正在吃东西”的信息。

Animal 类定义好后，下一步对 Animal 类进行实例化，即将 Animal 类作为一个实例载入所需要的视图中，代码如下：

```
let animal = new Animal();
```

ArkTS 中的类的实例化与其他语言实例化的方法基本类似，唯一不同的是需要添加 new 关键字，这是由 ArkTS 语法结构决定的，为了使在调用类的构造函数时能明确指向该类。由于在其他语言后续的迭代中慢慢地去掉了 new 关键字，所以此处仅作为了解，不做过多深入讲解。

在上述代码中，将 Animal 类赋值到 animal 的参数中，animal 就相当于 Animal 类的一个实例或对象，可以直接对 animal 进行操作，相当于对 Animal 类进行操作，代码如下：

```
//第 3 章/Index.ets
function showDetail() {
      let animal = new Animal();
      animal.name = "Peppa Pig";
      animal.eat();
}
```

在上述代码中，声明了一个函数 showDetail，在调用该函数时，首先创建了 Animal 类的实例 animal，然后对 Animal 类的属性进行赋值，最后调用 Animal 类中的 eat()函数打印赋值后的结果。

3.5.2 类的继承

可以创建很多类来描述事物的本质，而事物与事物之间常常会存在联系，或者一个事物是另一个事物的细化描述。面向对象编程的另一个重要概念是类的继承，通过继承，一个类可以从另一个类获取属性和方法。这样做不仅可以复用代码，减少重复，更重要的是它能够帮助组织和结构化代码，使代码的逻辑更加清晰，易于理解和维护。

假设有一个基本的 Animal 类，它有通用的属性如 name 和方法如 eat()。如果想描述更具体的动物，例如 Dog 和 Cat，这些动物都是动物的一种，但它们又有各自特有的行为。此时，可以让 Dog 和 Cat 类继承自 Animal 类，代码如下：

```
//第 3 章/Index.ets
class Animal {
      name = "";
      eat() {
      console.log(`${this.name} 正在吃东西`);
      }
}

class Cat extends Animal {
      hobby() {
      console.log(`${this.name} 喜欢抓猫抓板`);
      }
}
```

在上述代码中，创建了一个类 Cat，并使用 extends 关键字来实现继承 Animal 类的功能。实现类的继承逻辑后，Cat 类可以获得 Animal 类中的所有属性和方法。除此之外，Cat 类自己还定义了一种方法 hobby，调用该方法可以使用模板字符串 ${this.name}动态地输出猫的爱好——抓猫抓板。

下一步，可以对 Cat 类进行实例化，和实例化 Animal 类类似，代码如下：

```
//第 3 章/Index.ets
function showCatDetail() {
      let cat = new Cat();
      cat.name = "Tom";
      cat.eat();
      cat.hobby();
}
```

在上述代码中，对 Cat 类进行实例化，生成一个 cat 对象。之后，将 name 属性赋值为 Tom，并调用 eat()和 hobby()方法。执行时会输出“Tom 正在吃东西”“Tom 喜欢抓猫抓板”内容。

通过类的继承，面向对象编程能够更好地模拟现实世界中的对象关系，促进代码的模块化、重用性和扩展性。

3.6　本章小结

语法是编程语言与现实世界的抽象桥梁，其核心在于通过规则描述逻辑与数据交互。本章围绕 ArkTS 核心语法展开，涵盖参数声明、函数定义、条件判断、循环语句及面向对象编程。

对于初学者而言，掌握基础语法是突破编程门槛的关键。参数类型推导、函数封装、条件分支控制及类的继承机制均为实际开发中的高频工具。通过示例工程 LearnArkTS，读者可逐步理解 ArkTS 的设计哲学，并在后续复杂项目中灵活应用。

面向对象编程强调代码的模块化与复用性，而声明式语法则进一步简化界面开发。建议结合实践项目巩固本章内容，以提升开发效率与代码可维护性。