# 29｜面向对象编程第1步：先把基础搭好

2021-10-20 宫文学

《手把手带你写一门编程语言》 课程介绍 >

**讲述：宫文学**

时长 14:11 大小 12.99M

你好，我是宫文学。

到目前为止，我们的语言已经简单支持了 number 类型、string 类型和数组。现在，我们终于要来实现期待已久的面向对象功能了。

在我们的课程中，为了实现编译器的功能，我们使用了大量自定义的类。最典型的就是各种 AST 节点，它们都有共同的基类，然后各自又有自己属性或方法。这就是 TypeScript 面向对象特性最直观的体现。

面向对象特性是一个比较大的体系，涉及了很多知识点。我们会花两节课的时间，实现其中最关键的那些技术点，比如声明自定义类、创建对象、访问对象的属性和方法，以及对象的继承和多态，等等，让你理解面向对象的基础原理。

首先，我们仍然从编译器的前端部分改起，让它支持面向对象特性的语法和语义处理工作。

## 修改编译器前端

**首先是对语法的增强。**我们还是先来看一个例子，通过这个例子看看，我们到底需要增加哪些语法特性：

复制代码

```
class Mammal{
    weight:number;
    color:string;
    constructor(weight:number, color:string){
        this.weight = weight;
        this.color = color;
    }
    speak(){
        println("Hello!");
    }
}

let mammal = new Mammal(20,"white");
println(mammal.color);
println(mammal.weight);
println(mammal.speak);
```

在这个例子中，我们声明了一个 class，Mammal。这个类描述了哺乳动物的一些基础属性，包括它的体重 weight、颜色 color。它还提供了哺乳动物的一些行为特征，比如提供了一个 speak 方法。

Mammal 类还有一个特殊的方法，叫做构造方法。通过调用构造方法，可以创建类的实例，也就是对象。然后，我们可以访问对象的属性和方法。

其实 TypeScript 的类还有很多特性，包括私有成员、静态成员等等。这里我们还是先考虑一个最小的特性集合，先让语言支持最基础的类和对象特性。

看看这个示例程序，我们能总结出多个需要增强的语法点，包括类的声明、调用类的构造方法，this 关键字，以及通过点符号来引用对象的属性和方法。

我们首先看看类的声明。我们提供了下面这些语法规则，来支持类的声明：

```
classDecl : Class Identifier classTail ;
classTail :  '{' classElement* '}' ;
classElement : constructorDecl| propertyMemberDecl;
constructorDecl : Constructor '(' parameterList? ')' '{' functionBody '}' ;
propertyMemberDecl : Identifier typeAnnotation? ('=' expression)? ';'
                   | Identifier callSignature  '{' functionBody '}' ;
```

这些规则看似很多，但其实解析起来并不复杂。并且，很多基础成分在我们之前的函数声明的语法结构中都有了，比如参数列表（parameterList）、函数签名（callSignature）、函数体（functionBody）等等，所以这会给我们节省很多工作量。

第二，我们再看看如何调用类的构造方法来创建对象，比如“new Mammal(20,“white”)”这个表达式。

相关的语法规则如下：

```
primaryLeft: Identifier | functionCall | constructorCall |其他表达式
```

第三，在构造方法里，我们可以使用一个特殊的 this 关键字。这个关键字被解析以后，也会形成一个表达式，就叫做 This 表达式。

```
primaryLeft: Identifier | functionCall | constructorCall | This | 其他表达式
```

最后，我们需要用点符号来引用对象的属性和方法，比如用 mammal.weight 代表 mammal 对象的 weight 属性，这些都是我们平时很习惯使用的语法。

点号表达式跟上一节课的下标表达式一样，都是在别的表达式后面加个后缀，所以语法规则可以写成这样：

```
primary:  primaryLeft ('[' expression ']') | '.' expression)* ;
```

复制代码

好了，语法规则就是这些。

对应着这些语法规则，我们也需要增加一些 AST 节点，包括 ClassDecl、ConstructorDecl、PropertyDecl、MethodDecl、ConstructorCall、ThisExp 和 DotExp 等。

语法分析和 AST 节点的参考实现，你可以看看 parser.ts和 ast.ts。

**接下来，你肯定会猜到，我们还需要做一些语义分析的工作，包括再一次增强类型体系，以及符号表、引用消解等方面的工作，还要做一些必要的语义检查。**

在示例程序中，我们每次使用 class 声明一个新的类，实际上就是创建了一个自定义的类型。接下来，我们就可以用这个类型来声明变量，并进行类型检查。

这就给我们的语义分析增加了一些工作量，这个工作叫做类型消解。也就是说，当我们每次使用一个自定类型的时候，要知道这个类型是在哪里声明的。这就不像 number 和 string 这样的内置的类型，我们每次见到它们都可以自动识别出来。所以我们要多做一点工作，建立类型的声明和使用之间的关系。这种消解工作跟变量的消解、函数的消解是差不多的，可以尽量复用之前的算法。

在类型消解之后，我们就可以利用类的声明信息来进行类型检查了。比如，你可以在程序里用点符号访问 weight 属性、color 属性和 speak() 方法。但如果访问了没有定义过的属性或者方法，那就要报语义分析的错误。

除了这些，针对类的语义分析我们还有一些其他工作。比如，当我们创建对象的时候，对象的数据成员必须被正确地初始化。当然，那些可以取值为 undefined 的属性除外。所以，你需要使用我们第 22 节课学过的赋值分析技术，来完成这项工作。

好了，完成了语法分析和语义分析以后，编译器前端部分的工作就基本完成了。你可以运行我们的解析器，输出示例程序的 AST 看看。我把截屏放在了文稿里，并做了一些标注，方便你熟悉类的声明和类成员访问等语法对应的 AST 结构。

```
Prog
    ClassDecl Mammal
        ClassBody
            VariableStatement
                PropertyDecl weight:number
                    Number
                no initialization.
            VariableStatement
                PropertyDecl color:string
                    String
                no initialization.
            ConstructorDecl
                Return type: void
                    ParamList:
                        VariableDecl weight:number
                            Number
                        no initialization.
                        VariableDecl color:string
                            String
                        no initialization.
                    ExpressionStatement
                        Binary:Assign:number
                            DotExp
                              base
                                ThisExp
                              property
                                Variable: weight:number, LeftValue, resolved
                            Variable: weight:number, resolved
                    ExpressionStatement
                        Binary:Assign:string
                            DotExp
                              base
                                ThisExp
                              property
                                Variable: color:string, LeftValue, resolved
                            Variable: color:string, resolved
            FunctionDecl speak
                Return type: void
                    ExpressionStatement
                        FunctionCall (void)println, built-in
                            Hello!:string
```

完成了编译器前端的工作以后，我们接下来看看运行时方面需要做一些什么工作。首先，我们还是要升级一下 AST 解释器。

## 升级 AST 解释器

为了让 AST 解释器支持面向对象特性，我们需要做 4 个方面的工作，包括**创建对象、在栈桢里保存对象数据、通过点符号来引用对象的属性和方法，以及执行对象的方法**。

首先看创建对象的过程。你可以通过 new 关键字调用构造方法来创建对象。构造方法最重要的工作是初始化对象的属性。这里的具体实现，你可以参见visitConstructorCall方

法。

不过，对象的属性不仅仅是在构造函数里初始化的。其实，你在声明类的时候，就可以给属性带上初始化表达式。所以，实际对象的初始化过程，是首先使用这些初始化表达式，来给对象属性赋值，之后才会执行构造方法。

我们再看看第二个工作，在栈桢里保存对象数据。这里我用了一个简单的 Map 对象，来建立对象的属性和数据的映射关系。

接着是第三项任务，就是访问对象的属性。访问对象的属性需要借助点符号表达式。点符号左边的表达式，能够返回一个对象引用。在 AST 解释器里，这个对象引用和其他变量一样，都是一个 VarSymbol。基于这个 VarSymbol，程序就可以从栈桢里找到对象的数据，也就是我们前面说到的 Map 对象。你可以在这个 Map 里查找对象属性的值。

最后，是执行对象的方法。执行对象的方法跟执行普通函数其实差不多。主要的区别，**就是你必须给方法传递一个特殊的参数，也就是对象引用，具体来说就是一个 VarSymbol**。这样的话，你才可以在方法里用 this.weight 这样的表达式来访问对象的属性。这里的 this，就是传到方法里的对象引用。

好了，实现完这些机制以后，AST 解释器就顺利升级了。你可以用这个解释器运行一下前面的示例程序，输出结果如下图所示：


那么，升级完 AST 解释器以后，我们再进一步，尝试把我们这节课的示例程序编译成可执行程序。首先，我们仍然要设计一下对象的内存布局，并实现几个必要的内置函数。

## 内存布局和内置函数

在前面两节课，我们已经实现过了字符串和数组的内存结构。这些知识点我们今天仍然可以借鉴，降低我们这节课在设计和实现上的工作量。

我们用 PlayObject 来保存对象的数据。每个 PlayObject 跟 PlayString、PlayArray 等一样，也具备公共的对象头。在对象头之后，就是对象的属性数据。

目前我们可以存储 4 种类型的属性，包括 number 型、string 型、数组型和其他的自定义对象。它们都有一个共同的特点，就是**都需要在 PlayObject 中占据 8 个字节空间，用来保存数据或对象引用**。这是一个好消息，因为我们可以先简化对象的设计，不用考虑太多字节对齐等话题。

不过，如果你想了解字节对齐，你可以去参考一下 C 语言的结构体是如何安排每个字段在内存中的位置的。这些字段并不是一个挨着一个来存放的，相反，每个字段的其实地址，往往可以被 4 字节、8 字节等整除，这就导致字段之间可能存在空隙。这样做的原因，是让 CPU 在读取字节对齐的数据的时候速度更快，只需要在内部做一次读取操作就可以完成了。而读取不对齐的数据，CPU 在内部需要的读取操作，可就不止一次了。

这里我先不展开，你只需要知道内存布局设计上要考虑这个因素就行了。目前我们大可以不必担忧，因为我们存放的各种数据都是 8 字节大小，可以紧挨着排列，一点都不浪费空间，而且都是字节对齐的。

设计完内存布局之后，我们再实现一下内置函数。其中最主要的，当然就是在内存里创建对象的函数。

```
1 PlayObject* object_create_by_length(size_t length);
```

复制代码

我也用 C 语言写了一个class.c的测试程序，来测试创建对象、访问对象属性等功能。你可以用 make class 命令编译并运行一下看看。

好了，关于内存布局和内置函数，我们就讨论完毕了。接下来又到了最后一个环节：修改编译器后端。

## 修改编译器后端

在修改编译器后端的时候，我们需要把注意力放在两个方面：**访问对象的属性**和**调用对象的方法**。

我们先看看如何访问对象的属性。在上一节课，我们曾经实现过访问数组元素的功能。访问对象的属性和访问数组元素的原理，其实是一样的，关键点就是要正确地计算出内存地址。根据我们的内存布局设计，这其实就是在对象地址的基础上，加上一定的偏移量就可以了，实现起来很简单。

那我们再看看如何调用对象的方法。这倒是一个新的知识点，不过这跟调用函数有很多相似之处。我们对调用函数已经很熟悉了，而调用对象方法和它只有一个地方不同，就是方法的第一个参数，其实是对象引用。而原来方法声明中的第一个和第二个参数等等，则依次被往后移了一个位置，成为了第二个、第三个参数。其实，C++ 和 Java 等面向对象的语言，基本上也是用这样的方法传递对象引用的，让方法中的代码可以访问对象中的数据。

修改编译器后端的这些示例代码，我仍然放在了asm_x86-64.ts中，你可以参考一下。这里，你特别要注意看一下我是如何计算对象属性的内存地址，以及如何用参数机制给方法传递对象引用的。

## 课程小结

今天的主要内容就是这些了，我们再一起回顾一下这节课的重点：

在编译器前端方面，我们最近这几节课一直在迭代、增加一些语法规则，这一节课一下子又增加了不少。在我们课程的第一阶段，可能你需要花很多时间才能实现一小点语法规则。而现在，你可以大刀阔斧地快速实现很多语法规则了。是不是已经感受到了自己的技能提升了很多？

在这里，我也分享一点我的心得。在实现这些语法功能的时候，最重要的其实就是**设计出正确的语法规则**。一旦你能够清晰地写出语法规则，那么照着规则去实现语法分析程序就不是什么难事了。你可能不能一次写出完全正确的语法规则，这也没有关系，多尝试几次就好了。你会不断积累经验，直到对各种形式的语法都得心应手。

在内存布局方面，我们基本上沿袭了前几节课的设计。但关于字节对齐这个知识点，虽然我们当前的简化设计不会遇到字节对齐问题，但你仍然要了解它，以便以后升级我们的对

象设计，支持更多的数据类型，特别是小于 8 个字节的基础数据类型，比如 boolean 和 32 位的整型等。

最后，在编译器后端的实现上，重点是对象方法的调用机制。我们需要把对象引用作为第一个参数传递给方法。

好了，今天我们已经实现了基础的自定义对象功能。下一节课，我们会在这个基础上，增加面向对象编程最核心的一个特性，继承和多态功能，这会有助于加深你对面向对象的底层机制的理解。

## 思考题

TypeScript 中的几乎任何类型的数据，都可以用点符号来访问其内部的属性和方法。比如，我们可以访问字符串和数组的 length 属性，甚至也可以用一些方法调用 number 类型的数据。那么，基于今天课程中的知识点，你能不能思考一下，我们具体能如何实现上述这些功能呢？欢迎在留言区分享你的观点。

欢迎把这节课分享给更多感兴趣的朋友。我是宫文学，我们下节课见。

## 资源链接

这节课的示例代码都在这里！



上一篇 28 | 增加更丰富的类型第3步：支持数组

下一篇 30 | 面向对象编程第2步：剖析一些技术细节

## 精选留言 (1)

写留言

**奋斗的蜗牛**

2021-10-24

太赞了，面向对象的实现真是不简单

展开