



下载APP



## 30 | 面向对象编程第2步：剖析一些技术细节

2021-10-22 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 15:50 大小 14.52M



你好，我是宫文学。

在上一节课里，我们实现了基本的面向对象特性，包括声明类、创建对象、访问对象的属性和方法等等。

本来，我想马上进入对象的继承和多态的环节。但在准备示例程序的过程中，我发现有一些技术细节还是值得单独拿出来，和你剖析一下的，以免你在看代码的时候可能会抓不住关键点，不好消化。俗话说，魔鬼都在细节中。搞技术的时候，经常一个小细节就会成为拦路虎。



我想给你剖析的技术细节呢，主要是**语义分析**和**AST 解释器**方面的。通过研究这些技术细节，你会对面向对象的底层实现技术有更加细致的了解。

## 技术细节：语义分析

语义分析方面的技术细节包括：如何设计和保存 class 的符号、如何设计 class 对应的类型、如何给 This 表达式做引用消解、如何消解点符号表达式中变量等等。

**首先看看第一个问题，就是如何在符号表里保存 class 的符号。**

我们知道，符号表里存储的是我们自己在程序里声明出来的那些符号。在上一节课之前，我们在符号表里主要保存了两类数据：变量和函数。而 class 也是我们用程序声明出来的，所以也可以被纳入到符号表里保存。

你应该还记得，我们的符号表采用的是一种层次化的数据结构，也就是 Scope 的层层嵌套。而且，TypeScript 只允许在顶层的作用域中声明 class，不允许在 class 内部或函数内部嵌套声明 class，所以 class 的符号总是被保存在顶层的 Scope 中。

其实在 TypeScript 中，我们还可以在一个文件（或模块）里引用另一个文件里定义的类，这样你就能在当前文件里使用这些外部的类了。但我们其实并不是把外部类的全部代码都导入进来，而是只需要引入它们的符号就行了。在 class 符号里有这些类的描述信息，这些信息叫做元数据。元数据里会包括它都有哪些属性、哪些方法，分别都是什么类型的，等等。这些保存在符号里的这些信息，其实足够我们使用这个类了，我们不用去管这个类的实现细节。

你也可以对比一下 FunctionSymbol 的设计。FunctionSymbol 里会记录函数的名称、参数个数、参数类型和返回值类型。你通过这些信息就可以调用一个函数，完全不用管这个函数的实现细节，也不用区分它是内置函数，还是你自己写的函数，调用方式都是一样的。

**说完了 class 的符号设计和保存，我们再进入第二个技术点，讨论一下 class 的类型问题。**

我们说过，class 给我们提供了一个自定义类型的能力。那这个自定义的类型如何表达呢？

在前面的课程中，我们已经形成了自己的一套类型体系，用于进行类型计算。而在这个类型体系中，有一种类型叫做 NamedType。这些类型都有名称，并且还有父类型。我们用 NamedType 首先表示了 Number、String、Boolean 这些 TypeScript 官方规定的类

型，还用它来表示了 `Integer` 和 `Decimal` 这两个 `Number` 类型的子类型，这两个类型是我们自己设计的。

那其实 `NamedType` 就可以用来表示一个 `class` 的类型信息，以便参与类型计算。

这里你可能会提出一个问题：`class` 本身不就是类型吗？我们在 `ClassSymbol` 里已经保存了类的各种描述信息，为什么还要用到 `NamedType` 呢？

采用这样的设计有几个原因。首先，并不是所有的类型都是用 `class` 定义出来的。比如系统里有一些内置的类型。再比如，如果你用 `TypeScript` 调用其他语言编写的库，比如一些 `AI` 库，你可以把其他语言的类型映射成 `TypeScript` 语言的类型。所以说，类型的来源并不只有自定义的 `class`。

第二个原因是由类型计算的性质导致的。在我们目前的类型计算中，我们基本只用到了类型的名称和父子类型关系这两个信息，其他信息都没有用到，所以就不需要在类型体系中涉及。

不过，使用 `NamedType` 这种设计其实有个潜台词，就是我们类型系统是 `Norminal` 的类型系统。这是什么意思呢？`Norminal` 的意思是说，我们在做类型比较的时候，仅仅是通过类型的名称来确定两个类型是否相同，或者是否存在父子关系。与之对应的另一种类型系统是 `structural` 的，也就是只要两个类型拥有的方法是一致的，那就认为它们是相同的类型。像 `Java`、`C++` 这些语言，采用的是 `Nominal` 的类型系统，而 `TypeScript` 和 `Go` 等语言，采用的是 `Structural` 的类型系统。这个话题我们就不展开了，有兴趣你可以多查阅这方面的资料。

不过，为了简单，我们目前的实现暂且采用 `Norminal` 的类型，只通过名称来区分相互之间的关系。

**在分析完了 `class` 的符号和类型之后，我们再来看看它的用途。这就进入了第三个技术点，也就是如何消解 `This` 表达式。**

我们知道，`this` 表达式的使用场景，是在类的方法中指代当前对象的数据。那么它的类型是什么呢？在做引用消解的时候，应该让它引用哪个符号呢？

this 的类型，不用说，肯定就是指当前的这个 class 对应的类型，这个不会有疑问。

那它应该被关联到什么符号上呢？我们知道，当程序中出现某个变量名称或函数名称的时候，我们会把这些 AST 节点关联到符号表里的 VarSymbol 和 FunctionSymbol 上，this 当然也不会例外。this 在被用于程序中的时候，其用法跟普通的一个对象类型的变量是没有区别的。那我们是否应该在每个用到 this 的方法里，创建一个 this 变量呢？

这样当然可以，但其实也没有必要。因为每个函数都可能用到 this 关键字，所以如果在每个方法里都创建一个 this 变量有点啰嗦。我们只需要简单地把 this 变量跟 ClassSymbol 关联起来就行了，在使用的时候也没有什么不方便的。我们下面在讲 AST 解释器的实现机制里，会进一步看看如何通过 this 来访问对象数据。

### 接下来，我们再看看第四个技术点：对点符号表达式的引用消解。

在上一节课的示例程序中，我们可以通过 “this.weight”、“mammal.color”、“mammal.speak()” 这样的点符号表达式访问对象的属性和方法。

我们知道，在做引用消解的时候，需要把这里的 this、mammal、color、speak() 都关联到相应的符号上，这样我们就知道这些标识符都是在哪里声明的了。

不过，之前我们不是已经都做过引用消解了吗？为什么这里又要把点符号的引用消解单独拎出来分析呢？

这是因为，之前我们做变量和函数的引用消解的时候，只需要利用变量和函数的名称信息就行了。但在点符号这边，只依赖名称是不行的，还必须依赖类型信息。

比如，对于 mammal.color 这个表达式。我们在上下文里，很容易找到 mammal 是在哪里声明的。但 color 就不一样了。这个 color 是在哪里声明的呢？这个时候，你就必须知道 mammal 的类型，然后再找到 mammal 的定义。这样，你才能知道 mammal 是否有一个叫做 color 的属性。

那你可能说，这很简单呀，我们只需要先计算出每个表达式的类型，然后再做引用消解就可以了呀。

没那么简单。为什么呢？因为类型计算的时候，也需要用到引用消解的结果。比如在 `mammal.color` 中，如果你不知道 `mammal` 是在哪里声明的，就不能知道它的类型，那也就更没有办法去消解 `color` 属性了。

所以，在语义分析中，我们需要把类型计算和引用消解交叉着进行才行，不能分成单独的两个阶段。在 [《编译原理实战课》](#) 中，我曾经分析过 Java 的前端编译器的特点。这种多个分析工作穿插执行的情况，是 Java 编译器代码中最难以阅读和跟踪的部分，但你要知道这背后的原因。

我还给你提供了一个更复杂一点的例子，你可以先看一下：

[复制代码](#)

```
1 class Human{
2     swim(){
3         console.log("swim");
4     }
5 }
6
7 class Bird{
8     fly(){
9         console.log("fly");
10    }
11 }
12
13 function foo(animal:Human|Bird){
14     if (animal instanceof Human){
15         animal.swim();
16     }
17     else{
18         animal.fly();
19     }
20 }
```

这个例子里有 `Human` 和 `Bird` 两个类，`Human` 有 `swim()` 方法，而 `Bird` 有 `fly()` 方法。不过，我们可以声明一个变量 `animal`，是 `Human` 和 `Bird` 的联合类型。那么，你什么时候可以调用 `animal` 的 `swim()` 方法，什么时候可以调用它的 `fly()` 方法呢？这个时候你就要基于数据流分析方法，先进行类型的窄化，然后才能把 `swim()` 和 `fly()` 两个方法正确地消解。



好了，关于语义分析部分的一些技术点，我就先剖析到这里。接着我们看看 AST 解释器中的一些技术。


## 技术细节：Ast 解释器

实现 Ast 解释器的时候，我们也涉及了不少的技术细节，包括如何表示对象数据、对象数据在栈帧中的存储方式、如何以左值和右值的方式访问对象的属性等。

**首先我们看看如何表示对象的数据。**上一节课里，我们提到用一个 `Map<Symbol, any>` 来存储对象数据就行了。我们在类中声明的每一个属性，都对应着一个 `Symbol`，所以我们就可以用 `Symbol` 作为 `key`，来访问对象的数据。

其实，我们的栈帧也是这样设计的。每个栈帧也是一个 `Map<Symbol, any>`。你如果想访问哪个变量的数据，就把变量的 `Symbol` 作为 `key`，到 `Map` 里去查找就好了。

不过，如果只用一个 `Map` 来代表对象数据，数据的接收方可能不知道该数据是属于哪个类的，在实现一些功能的时候不方便。所以我们就专门设计了一个 `PlayObject` 对象，在对象里包含了 `ClassSymbol` 和对象数据两方面的信息，具体实现如下：

 复制代码

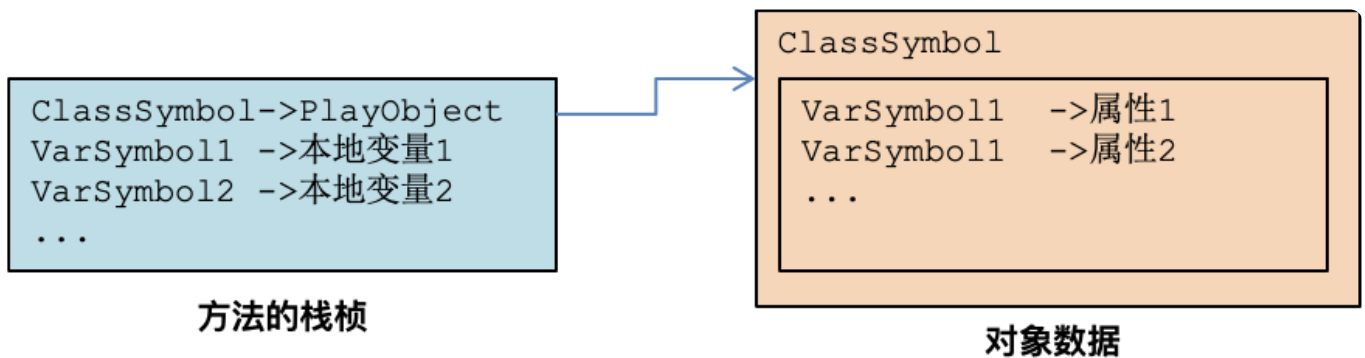
```
1 class PlayObject{
2     classSym:ClassSymbol;
3     data:Map<Symbol,any> = new Map();
4     constructor(classSym:ClassSymbol){
5         this.classSym = classSym;
6     }
7 }
```

### 那对象数据在栈帧里是如何保存的呢？

其实，每个方法都跟函数一样，会对应着一个栈帧。在方法里，如果我们用到了 `this` 关键字，那就可能会访问对象的属性或方法。

那我们就一定要在栈帧里放一个 `PlayObject` 对象。这个对象的 `key`，就是 `ClassSymbol`。正好，我们在前面让 `this` 表达式关联到了 `ClassSymbol` 上。所以，我们

使用 `this` 表达式就可以访问对象中的属性了。你看看下面的图，里面显示了栈帧和对象数据之间的关系，以及如何访问对象的属性。



那栈帧里的 `PlayObject` 对象是在什么时候被放到栈帧里的呢？其实是在调用构造方法和普通方法的时候。

在调用构造方法之前，我们首先要创建一个 `PlayObject` 对象，把它放到构造方法使用的栈帧里。在构造方法里，这样就可以用 `this` 来访问对象属性，并给这些属性赋予初始值了。另外，构造方法是没有返回值的。在调用构造方法之后，我们就把这个新创建的 `PlayObject` 当做返回值就好了。

在调用普通的对象方法的时候，比如用 `mammal.speak()` 对 `mammal` 求值的话，会返回一个 `PlayObject` 对象。然后我们把这个对象放在 `speak()` 方法的栈帧里就行了。

把 `PlayObject` 放在栈帧里，其实就相当于把 `PlayObject` 作为函数的第一个参数传到函数内部。总之，这样就能够在 `speak` 方法里使用 `this` 表达式了。

**最后，我们再看一下点符号表达式的左值和右值的使用场景。**在下面的示例程序中，第一句是给 `mammal.color` 赋值，所以我们需要一个左值。而第二句是获取 `mammal.color` 当前的值，所以是一个右值。

[复制代码](#)

```
1 mammal.color="yellow"; //左值
2 println(mammal.color); //右值
```

对于这两种场景，点符号表达式要分别返回左值和右值。在需要右值的时候，`mammal.color` 返回的是一个字符串。而在需要左值的时候，我们应该返回什么呢？

之前在处理本地变量的时候，我们已经学过，在需要左值的时候，直接返回变量的 Symbol 就好了。这样后续的赋值程序就可以把这个 Symbol 作为 Key 来修改栈帧中变量的值。

在 [28 讲](#) 中，在实现数组特性的时候，有时候我们需要修改某个数组元素的值。这个时候，我们就不能简单地用数组变量的 Symbol 来表达一个左值了，因为我们还需要知道数组元素的下标。所以那个时候，我们专门设计了另一个左值对象，叫做 `ArrayElementRef`。它里面甚至可以存放多个下标值，来引用多维数组中的某个元素。

[复制代码](#)

```
1 class ArrayElementRef{
2     varSym : VarSymbol; //数组的基础变量对应的Symbol
3     indices : number[]; //(多维)数组元素的下标。
4     constructor(varSym:VarSymbol, indices:number[]){
5         this.varSym = varSym;
6         this.indices = indices;
7     }
8 }
```

用于访问对象属性的左值，其实也可以采用类似的设计。这个类的名字叫做 `ObjectPropertyRef`，意思是这是对一个对象的属性的引用，里面有 `PlayObject` 对象，还有被访问的属性的 Symbol。基于这样一个左值，我们就可以修改对象的属性了。

[复制代码](#)

```
1 class ObjectPropertyRef{
2     object: PlayObject;
3     prop:Symbol;
4     constructor(object:PlayObject, prop:Symbol){
5         this.object = object;
6         this.prop = prop;
7     }
8 }
```

## 课程小结

今天的内容就是这些。通过这节课分享的一些技术实现细节，我希望你能记住几个关键点。



首先，在语义分析方面，我们需要对 class 建立符号，并存到符号表里。class 的符号应该包含足够的描述信息，包括名称，以及属性和方法的描述。为了进行类型计算，我们还要把 class 符号关联到一个 NamedType 对象中。这种类型计算方式的设计思路，是基于 Nominal 的类型系统而来的。

在支持点符号表达式以后，我们的引用消解和类型计算需要交错起来进行，这会导致语义分析程序变得复杂。你在查看各种编译器的源代码的时候，也可以多关注它们在这方面是如何实现的。

第二，在 AST 解释器的实现机制上，你的脑海里需要对栈帧有一个清晰的图像。在对象方法的栈帧里，我们一定会放一个 PlayObject 对象数据，这样就可以用 this 来访问对象的属性了。在访问对象属性时，又要分为左值和右值的情况。对于左值，我们要设计一种数据结构，清晰地表达出如何访问对象的属性。

最后，关于课程代码的学习，我还要再叮嘱你几句。

在课程起步篇的后半段和我们现在的进阶篇里，课程的示例代码的体量明显加大。并且，由于每节课示例代码都在迭代，你在阅读代码的时候可能会感觉到有一定的负担。

这里我想强调的是，编译器针对词法分析和语法分析这样功能的代码，往往大家的实现都差不多。因为这两部分的理论化是最强的，基本上你理解了理论就能写出差不多的代码来了。

而语义分析、编译器的后端等的代码，工程特点就比较强了，各个编译器的实现差异很大。你需要把握其中的关键技术点，就比如今天我们这节课分析的这些点。这样在具体实现上，你可以不用拘泥于哪种具体的方式。就比如，在这节课中，关于如何消解 this，以及如何把对象数据提供给方法，并通过 this 访问，其实可以有多种技术方案。你可以活学活用，只要把握住其中的关键点就可以了。

## 思考题

今天我们讲到了 class 的符号中包含的信息。那你能不能思考一下，这些符号是否需要在虚拟机或者可执行程序保存？保存这些信息有什么用途？你能不能结合你熟悉的语言来分享一下？

另外，今天我们提到的 Norminal 和 Structural 的类型系统，你在使用它们的时候有什么体会？如果我们想要实现 Sturctural 的类型系统，那应该如何设计？欢迎你在留言区分享观点。

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。

分享给需要的人，Ta订阅后你可得 **20 元现金奖励**

 生成海报并分享

 赞 0

 提建议

© 版权归极客邦科技所有，未经许可不得传播售卖。页面已增加防盗追踪，如有侵权极客邦将依法追究其法律责任。

上一篇 29 | 面向对象编程第1步：先把基础搭好

下一篇 31 | 面向对象编程第2步：支持继承和多态

1024 活动特惠

VIP 年卡直降 ¥2000

新课上线即解锁，享 365 天畅看全场

超值拿下 ¥999 



精选留言 (1)

 写留言

**罗乾林**

2021-10-22

class的符号信息，这些符号可以不在在虚拟机或者可执行程序中保存，如果保存了这些信息，可以让我们的语言更加灵活，如实现运行时的类加载，类自省等能力比如JAVA。不保存这些信息的有C/C++(c++有typeinfo不知道算不算)。

Norminal和Structural的类型系统，感觉Structural的类型系统更加灵活动态性更好，在编...

展开 ∨

