

38 | 元编程：一边写程序，一边写语言

2019-11-20 宫文学 来自北京

《编译原理之美》



今天，我再带你讨论一个很有趣的话题：元编程。把这个话题放在这一篇的压轴位置，也暗示了这个话题的重要性。

我估计很多同学会觉得元编程（Meta Programming）很神秘。编程，你不陌生，但什么是元编程呢？

元编程是这样一种技术：你可以让计算机程序来操纵程序，也就是说，用程序修改或生成程序。另一种说法是，具有元编程能力的语言，能够把程序当做数据来处理，从而让程序产生程序。

而元编程也有传统编程所不具备的好处：比如，可以用更简单的编码来实现某个功能，以及可以按需产生、完成某个功能的代码，从而让系统更有灵活性。


某种意义上，元编程让程序员拥有了语言设计者的一些权力。是不是很酷？你甚至可以说，普通程序员自己写程序，文艺程序员让程序写程序。

那么本节课，我会带你通过实际的例子，详细地来理解什么是元编程，然后探讨带有元编程能力的语言的特性，以及与编译技术的关系。通过这样的讨论，我希望你能理解元编程的思维，并利用编译技术和元编程的思维，提升自己的编程水平。

从 Lisp 语言了解元编程

说起元编程，追溯源头，应该追到 Lisp 语言。这门语言其实没有复杂的语法结构，仅有的语法结构就是一个个函数嵌套的调用，就像下面的表达式，其中 “+” 和 “*” 也是函数，并不是其他语言中的操作符：

```
1 (+ 2 (* 3 5)) //对2和3求和，这里+是一个函数，并不是操作符
```

 复制代码

你会发现，如果解析 Lisp 语言形成 AST，是特别简单的事情，基本上括号嵌套的结构，就是 AST 的树状结构（其实，你让 Antlr 打印输出 AST 的时候，它缺省就是按照 Lisp 的格式输出的，括号嵌套括号）。这也是 Lisp 容易支持元编程的根本原因，你实际上可以通过程序来生成，或修改 AST。

我采用了 Common Lisp 的一个实现，叫做 SBCL。在 macOS 下，你可以用 “brew install sbcl” 来安装它；而在 Windows 平台，你需要到 sbcl.org 去下载安装。在命令行输入 sbcl，就可以进入它的 REPL，你可以试着输入刚才的代码运行一下。

在 Lisp 中，你可以把 (+ 2 (* 3 5)) 看做一段代码，也可以看做是一个列表数据。所以，你可以生成这样一组数据，然后作为代码执行。**这就是 Lisp 的宏功能。**

我们通过一个例子来看一下，宏跟普通的函数有什么不同。下面两段代码分别是用 Java 和 Common Lisp 写的，都是求一组数据的最大值。

Java 版本：

[复制代码](#)

```
1 public static int max(int[] numbers) {
2     int rtn = numbers[0];
3     for (int i = 1; i < numbers.length; i++){
4         if (numbers[i] > rtn)
5             rtn = numbers[i];
6     }
7     return rtn;
8 }
```

Common Lisp 版本:

[复制代码](#)

```
1 (defun mymax1 (list)
2   (let ((rtn (first list)))           ;让rtn等于list的第一个元素
3     (do ((i 1 (1+ i)))                ;做一个循环, 让i从1开始, 每次加1
4       ((>= i (length list)) rtn)      ;循环终止条件: i>=list的长度
5       (when (> (nth i list) rtn)      ;如果list的第i个元素 > rtn
6         (setf rtn (nth i list))))))  ;让rtn等于list的第i个元素
```

那么, 如果写一个函数, 去求一组数据的最小值, 你该怎么做呢? 采用普通的编程方法, 你会重写一个函数, 里面大部分代码都跟求最大值的代码一样, 只是把其中的一个 ">" 改为 "<"。


这样的话, 代码很冗余。那么, 能不能实现代码复用呢? 这一点, 用普通的编程方法是做不到的, 你需要利用元编程技术。我们用 Lisp 的宏来实现一下:

[复制代码](#)

```
1 (defmacro maxmin(list pred)
2   `(let ((rtn (first ,list)))
3     (do ((i 1 (1+ i)))
4       ((>= i (length ,list)) rtn)
5       (when (,pred (nth i ,list) rtn)
6         (setf rtn (nth i ,list))))))
7
8 (defun mymax2 (list)
9   (maxmin list >))
10
11 (defun mymin2 (list)
```

```
12      (maxmin list <))
```

在宏中，到底使用 “>” 还是使用 “<”，是可以作为参数传入的。你可以看一下函数 `mymax2` 和 `mymin2` 的定义。这样，宏展开后，就形成了不同的代码。你可以敲入下面的命令，显示一下宏展开后的效果（跟我们前面定义的 `mymax1` 函数是完全一样的）。

 复制代码

```
1 (macroexpand-1 '(maxmin list >))
```

在 Lisp 运行时，会先进行宏展开，然后再编译或解释执行所生成的代码。通过这个例子，你是否理解了“用程序写程序”的含义呢？这种元编程技术用好了以后，会让代码特别精简，产生很多神奇的效果。

初步了解了元编程的含义之后，你可能会问，我们毕竟不熟悉 Lisp 语言，目前那些常见的语言有没有元编程机制呢？我们又该如何加以利用呢？

不同语言的元编程机制

首先，我们回到元编程的定义上来。比较狭义的定义认为，一门语言要像 Lisp 那样，要能够把程序当做数据来操作，这样才算是具备元编程的能力。

但是，你学过编译原理就知道，在 CPU 眼里，程序本来就是数据。

我们在 [🔗 34 讲](#)，曾经直接把二进制机器码放到内存，然后作为函数调用执行。有一位同学在评论区留言说，这看上去就是把程序当数据处理。在 [🔗 32 讲](#)中，我们也曾生成字节码，并动态加载进 JVM 中运行。这也是把程序当数据处理。

实际上，整个课程，都是在把程序当做数据来处理。你先把文本形式的代码变成 Token，再变成 AST，然后是 IR，最后是汇编代码和机器代码。所以，有的研究者认为，编写编译器、汇编器、解释器、链接器、加载器、调试器，实际上都是在做元编程的工作，你可以参考一下 [🔗 这篇文章](#)。

从这里，你应该得到一个启示：学习汇编技术以后，你应该有更强的自信，去发掘你所采用的语言的元编程能力，从而去实现一些高级的功能。

当然了，通常我们说某个语言的元编程能力，要求并不高，没必要都去实现一个编译器（当然，如果必须要实现，你还是能做到的），而是利用语言本身的特性来操纵程序。**这又分为两个级别：**

如果一门语言写的程序，能够报告它自身的信息，这叫做自省（introspection）。

如果能够再进一步，操纵它自身，那就更高级一些，这叫做反射（reflection）。

那么你常见的语言，都具备哪些元编程能力呢？

1. JavaScript

从代码的可操纵性来看，JavaScript 是很灵活的，可以给高水平的程序员，留下充分发挥的空间。JavaScript 的对象就跟一个字典差不多，你可以随时给它添加或修改某个属性，你也可以通过拼接字符串，形成一段 JavaScript 代码，然后再用 `eval()` 解释执行。JavaScript 还提供了一个 `Reflect` 对象，帮你更方便地操纵对象。

实际上，JavaScript 被认为是继承了 Lisp 衣钵的几门语言之一，因为 JavaScript 的对象确实就是个可以随意修改的数据结构。这也难怪有人用 JavaScript，实现了很多优秀的框架，比如 React 和 Vue。

2. Java

从元编程的定义来看，Java 的反射机制就算是一种元编程机制。你可以查询一个对象的属性和方法，你也可以用程序按需生成对象和方法来处理某些问题。

我们 [🔗 32 讲](#) 中的字节码生成技术，也是 Java 可以采用的元编程技术。你再配合上注解机制或者配置文件，就能实现类似 Spring 的功能。可以说，Spring 是采用了元编程技术的典范。

3. Clojure

Clojure 语言是在 JVM 上，运行的一个现代版本的 Lisp 语言，所以它也继承了 Lisp 的元编程机制。

4. Ruby

喜欢 Ruby 语言的人很多，一个重要原因在于 Ruby 的元编程能力。而 Ruby 也声称自己继承了 Lisp 语言的精髓。其实，它的元编程能力表现在，能够在运行时，随时修改对象的属性和方法。虽然实现方式不一样，但原理和 JavaScript 其实是很像的。

元编程技术使 Ruby 语言能够以很简单的方式快速实现功能，但因为 Ruby 过于动态，所以编译优化比较困难，性能比较差。Twitter 最早是基于 Ruby 写的，但后来由于性能原因改成了 Java。同样是动态性很强的语言，JavaScript 在浏览器里使用普遍，厂商们做了大量的投入进行优化，因此，JavaScript 在大部分情况下的性能，比 Ruby 高很多，有的 [测试用例](#)会高 50 倍以上。所以近几年，Ruby 的流行度在下降。**这也侧面说明了编译器后端技术的重要性。**

5. C++ 语言

C++ 语言也有元编程功能，最主要的就是模板（Template）技术。

C++ 标准库里的很多工具，都是用模板技术来写的，这部分功能叫做 STL（Standard Template Library），其中常用的是 vector、map、set 这些集合类。**它们的特点是**，都能保存各种类型的数据。

看上去像是 Java 的泛型，如 `vector< T >`，但 C++ 和 Java 的实现机制是非常不同的。我们在 [35 讲](#)中曾经提到 Java 的泛型，指出 Java 的泛型只是做了类型检查，实际上保存的都是 Object 对象的引用，`List< Integer >` 和 `List< String >` 对应的字节码是相同的。

C++ 的模板则不一样。它像 Lisp 的宏一样，能够在编译期展开，生成 C++ 代码再编译。`vector< double >` 和 `vector< long >` 所生成的源代码是不同的，编译后的目标代码，当然也是不同的。

常见语言的元编程特性，你现在已经有所了解了。但是，关于是否应该用元编程的方法写程序，以及如何利用元编程方法，却存在一些争议。

是否该使用元编程技术？

我们看到，很多支持元编程技术的语言，都声称继承了 Lisp 的设计思想。Lisp 语言也一致被认为是编程高手应该去使用的语言。可有一个悖论是，Lisp 语言至今也还很小众。

Lisp 语言的倡导者之一，Paul Graham，在互联网发展的早期，曾经用 Lisp 编写了一个互联网软件 Viaweb，后来被 Yahoo 收购。但 Yahoo 收购以后，就用 C++ 重新改写了。**问题是：**如果 Lisp 这么优秀，为什么会被替换掉呢？

所以，一方面，Lisp 受到很多极客的推崇，比如自由软件的领袖 Richard Stallman 就是 Lisp 的倡导者，他写的 Emacs 编辑器就采用了 Lisp 来自动实现很多功能。

另一方面，Lisp 却没有成为被大多数程序员所接受的语言。这该怎么解释呢？难道普通程序员不聪明，以至于没有办法掌握宏？进一步说，我们应该怎样看待元编程这种酷炫的技术呢？该不该用 Lisp 的宏那样的机制来编程呢？

程序员的圈子里，争论这个问题，争论了很多年。**我比较赞同的一个看法是这样的：**首先，像 Lisp 宏这样的元编程是很有用的，你可以用宏写出非常巧妙的库和框架，来给普通的程序员来用。但一个人写的宏对另外的人来说，确实是比较难懂、难维护的。从软件开发管理的角度看，难以维护的宏不是好事情。

所以，我的结论是：

首先，元编程还是比较高级的程序员的工作，就像比较高级的程序员才能写编译器一样。元编程其实比写编译器简单，但还是比一般的编程要难。

第二，如果你要用到元编程技术，最好所提供的软件是容易学习、维护良好的，就像 React、Vue 和 Spring 那样。这样，其他程序员只需要使用就行了，不必承担维护的职责。

其实，我们学编译技术也是一样的。你不能指望公司或者项目组的每个人，都用编译技术写一个 DSL 或者写一个工具。毕竟维护这样的代码有一定的门槛，使用这些工具的人也有一定的学习成本。我曾经看到社区里有工程师抱怨，某国外大的互联网公司里面 DSL 泛滥，新加入的成员学习成本很高。所以，一个 DSL 也好、一套类库也好，必须提供的价值远远大于学习成本，才能被广泛接受。

为了降低使用者的学习成本，框架、工具的接口设计应该非常友好。**怎样才算是友好呢？**我们可以借鉴特定领域语言（DSL）的思路。

发明自己的特定领域语言（DSL）

框架和工具的设计者，为了解决某一个特定领域的问题，需要仔细设计自己的接口。好的接口设计是对领域问题的抽象，并通过这种抽象屏蔽了底层的技术细节。这跟上一讲我们提到语言设计的抽象原则是一样的。这样的面向领域的、设计良好的接口，很多情况下都表现为 DSL，例如 React 的 JSX、Spring 的配置文件或注解。

DSL 既然叫做语言，那么就应该具备语言设计的特征：通过简单的上层语义，屏蔽下层的技术细节，降低认知成本。

我很早以前就在 BPM 领域工作。像 JBPM 这样的开源软件，都提供了一个定义流程的模板，也就是 DSL。**这种 DSL 的优点是：**你只需要了解与业务流程这个领域有关的知识，就可以定义一个流程，不需要知道流程实现的细节，学习成本很低。

🔗 15 讲的报表工具的例子，也提供了一个报表模板的参考设计，这也是一个 DSL。使用这个 DSL 的人也不需要了解报表实现的细节，也是符合抽象原则的。

我们在日常工作中，还会发现很多这样的需求。你会想，如果有一门专门干这个事情的 DSL 就好了。比如，前两年我参与过一个儿童教育项目，教师需要一些带有动画的课件。如果要让一个卡通人物动起来，动画设计人员需要做很多繁琐的工作。当时就想，如果有一个语言，能够驱动这些卡通人物，让它做什么动作就做什么动作，屏蔽底层的技术复杂性，那么那些老师们就可以自己做动画了，充分发挥自己的创造力，而不需要求助于专门的技术人员。

当然，要实现这种 DSL，有时候可以借助语言自带的元编程能力，就像 React 用 JavaScript 就能实现自己的 DSL。但如果 DSL 的难度比较高，那还是要实现一个编译器，这可能就是终极的元编程技能了吧！

课程小结

本节课，我带你了解了元编程这个话题，并把它跟编译原理联系在一起，做了一些讨论。学习编译原理的人，某种意义上都是语言的设计者。而元编程，也是让程序员具有语言设计者的能力。所以，你可以利用自己关于编译的知识，来深入掌握自己所采用的语言的元编程能力。

我希望你能记住几个要点：

元编程是指用程序操纵程序的能力，也就是用程序修改或者生成程序。也有人用另外的表述方式，认为具有元编程能力的语言，能够把程序当做数据来处理，典型的代表是 Lisp 语言。

编译技术的本质就是把程序当做数据处理，所以你可以用编译技术的视角考察各种语言是如何实现元编程的。

采用元编程技术，要保证所实现的软件是容易学习、维护良好的。

好的 DSL 能够抽象出领域的特点，不需要使用者关心下层的技术细节。DSL 可以用元编程技术实现，也可以用我们本课程的编译技术实现。

一课一思

你之前了解过元编程技术吗？你曾经用元编程技术解决过什么问题呢？欢迎在留言区分享。

最后，感谢你的阅读，如果这篇文章让你有所收获，也欢迎你将它分享给更多的朋友。

© 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。



brian

2020-05-25

老师能不能出个专栏，从编译原理的视角来解析串讲，沿着历史时间线上的主流编程语言的
语言特性

作者回复: 嗯，回顾计算机语言和相关编译技术的演进历程，这个可以有。

不仅仅是讲技术，我觉得讲讲技术背后的人，那些做出重要贡献的人的故事，以及当时做那些工作的
背景，我觉得都挺有意思的。

是否能写成专栏我不确定，但这确实是一个好的题目。

共 3 条评论 >



4



Calios

2020-02-13

go的reflect库、objective-c的runtime，看起来提供的都是元编程的能力。听老师讲，才有一
种茅塞顿开的感觉。 🙏

作者回复: Great!

元编程能够帮助你编写通用性更强的程序。有了元编程的思维，你的工具箱里又多了一把利器。



4



William Ning

2019-11-21

拉下好多了

作者回复: 没事，课程一直在，老师一直在。

共 2 条评论 >



2



Milittle

2019-11-21

老师有没有推荐学习编译的blog，就是那种大神的博客 从入门到精通？谢谢~

作者回复: 我曾经关注过国内国外的几个极客，有几个写得很不错。我整理一下，放到加餐里！

共 2 条评论 >



2



陌兮



2022-06-11

之前从来没有接触过元编程的概念。现在听老师一说，感觉清晰多了。原来我们日常中也会经常使用到。比如自定义配置规则，自定义注解解析等，都使用到了元编程



if...else... 

2021-10-28

哈哈，长见识了



曾泽浩

2021-04-11

元编程和函数编程是一回事吗？

作者回复：不是一回事。

元编程，Metaprogramming，一旦你看到Meta，意味着你的工作的抽象层次更高。你还可以看看《编译原理实战课》的第36讲，会把Meta这个概念介绍得更深。

而函数式编程，是一种编程范式，相对于命令式编程而言的。

共 2 条评论 >



pencilCool

2020-09-24

本节课的主要收获

- 1 反射和自省概念的区别
- 2 javascript 也可以玩元编程
- 3 Clojure 是现代版的lisp
- 4 JavaScript 的性能比 ruby 高
- 5 c++ 和 java 范型的区别
- 6 Lisp 不好维护代码： 别人宏定义有学习成本

作者回复：Great!



pencilCool

2020-09-24

以前看过 ruby 元编程那本书，感觉ruby 写dsl 好自然呀。

作者回复: 是的。很多程序员喜欢ruby的原因，都在于ruby的元编程能力赋予了程序员更大的发挥空间。

在第二季《编译原理实战课》中，对元编程的本质和实现机制有更深入的介绍，可以去看看。



刘強

2020-04-03

返回来再看看，lisp宏那儿有个小错误吧，宏应该返回列表，前面应该加quote了吧

作者回复: 是的。谢谢你的细心。

文稿发布到app的时候，可能不好处理那个quote符号。我已经请编辑同学处理。

我放了一份在github的上，可供参考：<https://github.com/RichardGong/PlayWithCompiler/blob/master/lab/38-meta/meta.lsp>



拉欧

2019-12-13

lisp以前了解过，感觉宏这种东西确实太小众了，函数逻辑已经很复杂难维护了，再加上宏，画美不看。。。

作者回复: 确实，语言设计要考虑学习成本。

