



下载APP



课前热身 | 开始学习之前我们要准备什么？

2021-08-09 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 15:49 大小 14.49M



你好！我是宫文学，欢迎来到《手把手带你写一门编程语言》的课程。

其实，你从课程题目就可以看出，我们这个课强调动手实践。所以在这一节课，我要给你介绍一下我们这个课程示例代码所采用的计算机语言，以及相关编程环境的搭建。这样，会方便你阅读、运行和修改课程的示例代码。

对于课程里用到的汇编语言、编译原理知识，如果你之前没有相关的经验，也不要担心。我会介绍一下我们这方面的设计思路，保证你通过这个课程会更快、更扎实地掌握它们



通过这篇导读，你会对课程里用到的语言、工具、技术心里有数，以便更好地开启你的学习之旅。

好，我们先从使用的计算机语言和环境说起。

怎么快速上手 TypeScript 语言

我们这个课程的目标呢，是要实现 TypeScript 的编译器和各种运行时。既然如此，那么我就尽可能地用 TypeScript 来实现这个目标。

虽然我们这个课程主体的代码都是用 TypeScript 写的，但我正式使用 TypeScript 其实是从 2021 年 5 月份开始，也就是我开始准备这个课的时间。

我知道你肯定会问：用几个月的时间，既要了解 TypeScript，又要用 TypeScript 写自己的编译器，是不是太不靠谱了？当然，你可能也是因为要学习这门课程，第一次使用 TypeScript，所以我就分享一下自己的一些经验。

第一，使用它！

我一直觉得，真正的语言学习开始于你使用它的那一刻。否则，你就是一直看这门语言的资料，也只能留下个大概印象，而且很快就会忘掉，只有动手使用，才会形成肌肉记忆。比如，现在我一写 for 循环，手指不自觉地打出 “for (let i = ...)” 或 “for (let x of ...)” 开头，这就是形成肌肉记忆了。

第二，看资料！


说实在的，现在学习计算机语言实在是太方便了，各种资料应有尽有，又有很多热心同学在网上的分享，遇到什么需求一查就有，用过才会记住。

如果你让我推荐一本学习资料，我比较推荐流浪小猫写的开源电子书 [🔗 《TypeScript 入门教程》](#)。这个作者可能比较懂我想看什么，提供的内容会到点上。比如，我们做面向对象编程的时候，都关心该语言是否具备运行时的类型判断能力，因为这个功能几乎百分之百会被用到，而这个教程里就专门有 [🔗 类型断言](#) 的章节。

第三，靠经验直觉！


其实，很多同学都学过多门语言，那么学一门新语言的速度就会很快。有经验的程序员会建立一些直觉，能够猜到一门语言可能具备什么特性。

在准备课程代码的时候，我有一次要编写一个类，代表汇编指令中的寄存器操作数。而在 x86 的汇编指令中，不同位数的寄存器的名称是不一样的，所以，我需要用成员变量 bits，来表示这个寄存器的位数。这个时候，我想当然地写出了下面的代码：

 复制代码

```
1 class Register extends Operand{
2     bits:32|64 = 32; //寄存器的位数
3     ...
4 }
```

也就是位数只能取两个值，要么是 32，要么是 64，赋其他值给它都是错误的。

在使用这个写法的时候，我其实不是很确定是否可以用两个值的集合来描述变量的类型，因为在教程里提到  联合类型 (Union Types) 的时候，只是说了可以用两个类型的联合。我直觉上觉得也应该支持两个值的联合，因为如果我是 TypeScript 的作者，我可能不会忽视这种使用场景。我根据自己的猜测试了一下，然后就成功了。

如果上升到类型理论的高度，那么我们可以说，类型本来就是可以取的值的集合。但如果我们不上升到理论高度，仅凭直觉，其实也能去正确地使用类型。

说到老程序员的直觉，其实这门课程的一个重要目标，就是想帮你建立起更多的直觉，建立仅仅通过高级语言的语法表象就能看透其内部实现机制的能力，让计算机语言在你面前成为一个白盒子，从而让你能够更加自如地去支配不同的语言来为自己服务。

好了，了解了怎么上手以后，我们再来看看在这门课程中的 TypeScript 的环境配置问题。

TypeScript 的环境配置

我们这个课程关于 TypeScript 的环境配置主要包括这些：

1. 编译和运行环境：Node.js。

首先，我使用 Node.js 来编译和运行 TypeScript，所以你要先在自己的电脑上安装 Node.js，配置好相应的环境。这方面的资料很多，我就不提供链接了。

2. 安装和配置 TypeScript。

使用下面的命令，可以安装 TypeScript：

```
1 npm install typescript -g
```

[📄 复制代码](#)

之后，你可以用 git 命令下载示例代码：

```
1 git clone https://gitee.com/richard-gong/craft-a-language/tree/master
```

[📄 复制代码](#)

在用 git 下载了示例代码以后，需要你在示例代码的目录中运行下面这个命令，安装示例程序依赖的 node.js 中的一些包。安装完毕以后，会在 craft-a-language 目录中建立一个 node_modules 子目录：

```
1 npm i --save-dev @types/node
```

[📄 复制代码](#)

3.IDE：Visual Studio Code（简称 VS Code）。

我在课程里使用了 VS Code 作为 IDE，VS Code 缺省就支持 TypeScript 语言，毕竟这个 IDE 本身就是用 TypeScript 编写的。

而且，我们每一节课的代码，都会被放在我们 [📁 代码库](#) 里的一个单独的目录下，比如 01、02.....在每个目录下，都会有几个 json 文件，是 TypeScript 的工程配置文件。你只要在目录下输入 tsc，就会编译该工程的所有.ts 文件。

打开 tsconfig.json，你会看到我提供的一些配置项：

target 项是编译目标，这里我用的是 es6，因为 es6 具备了很多高级特性。

module 项是模块管理工具，我们选用的是 CommonJS。

另外，有“exclude”选项，是排除了一些.ts文件，不纳入编译范围。这些.ts文件都是以example开头，是我们自己课程的一些例子，给我们自己的编译器使用的，就不用tsc编译了。

4. 运行我们自己的语言：

我们自己的语言的入口，是play.ts，你可以通过下面这个命令来运行example.ts程序：

```
1 node play example.ts
```

[复制代码](#)

C 语言相关的功能和配置

我们这个课程的部分内容还使用了C语言，主要用来实现一些运行时的功能，包括：

实现了一个运行字节码的虚拟机：

我们首先用TypeScript实现了一个虚拟机，然后用C语言又实现了一版。因为现实世界的很多虚拟机，都是用C/C++实现的，所以我们要体会一下系统级的语言来实现虚拟机有什么不同，特别是性能上会有什么提升。同时，也能体会到字节码带来的跨平台特性：针对相同的字节码，不同的虚拟机的运行结果是完全一样的。

实现了一些内置函数：

每门语言都有一些内置的函数，由于其功能比较底层，所以要用系统级的语言来实现。比如，我们在语言里提供了一个println()的内置函数，用来打印信息，方便程序调试。这个函数我是用C语言的标准库实现的，然后在生成的汇编代码中调用这个函数就行。

还有一些内置函数，叫做Intrinsics，它们不是给开发者使用的，而是被编译器所调用。比如，为了实现TypeScript对字符串数据类型的支持，我用C语言写了几个函数，分别用于创建字符串对象、返回字符串长度，还有进行字符串连接等功能。很多时候，语言的使用者意识不到这些内核函数的存在，但他们使用TypeScript本身的语法来处理字符串的时候，在运行时里就会调用这些内置函数。

实现了内存管理功能：

TypeScript 是面向对象的语言，在使用对象的过程中，需要内存管理功能来生成内存中的对象。在必要的时候，又会运行垃圾收集功能，回收不再被使用的对象，这些功能也是用 C 语言实现的。

我想，大部分同学对 C 语言应该还是比较熟悉的。虽然有很多同学在工作以后可能不太使用 C 语言，但很多在大学的时候都学习过，做过课程练习，所以阅读和编写课程中的示例代码，应该难度不大。C 语言的特点就是很简洁，所以也受到很多极客的欢迎。如果我用 C++ 来写这些代码，可能就会给你带来一些阅读障碍了。

我们这门课 C 语言的配置环境是这样的：

1. 编译器等工具链。

我平常工作采用的是 MacOS 系统，所以系统有自带的 clang 工具链。同时，你也可以使用 gcc，我在课程里使用的 clang 命令，其实换成 gcc 都能运行。所以，如果你用 Linux/Windows 系统做课程练习，就可以用 gcc 开发环境，Windows 系统可以参考 [这篇文档](#)。

2.IDE : Visual Studio Code。

如果你要编写和调试这些 C 语言的代码，仍然用 VS Code 就可以了。而且，我用起来已经比较习惯了，觉得没必要采用一个商用的 IDE。

这门课的 C 语言的代码都被我放在了 [代码库](#)里的 vm 和 rt 两个子目录下，一个是虚拟机的代码，一个是运行时的代码。在这两个目录下，都有一个.vscode 目录，里面有一些配置文件，告诉 VS Code 如何做编译和代码调试。

最后，我想说明一下，我在课程里使用 C 语言其实还有另一个重要的用途，就是观察同样的功能，用 C 语言生成的汇编代码是什么样子的，从而加以学习和模仿。根据我的经验，这是最快熟悉汇编语言的途径。

那我们接下来就说说汇编语言。

你是否需要懂汇编语言？

在这个课程里，我们会实现编译器的后端，也就是把 TypeScript 编译成汇编代码，从而生成二进制的可执行文件。所以，从这个意义上说，**我们需要了解汇编语言**。

但从现实角度，除了一些嵌入式开发、芯片开发、驱动开发、操作系统，还有编译器等领域的工程师，很少有人日常也使用汇编语言。就算使用，也很少人能够熟悉各种不同芯片的指令集。

所以，这个课程关于汇编语言的部分，不需要你提前学过相关课程。我会用一节课，给你介绍一下物理计算机的运行原理，这是使用汇编语言的背景知识；然后再用一节课，介绍汇编语言的一些共性的知识，包括不同芯片的指令集都会有的一些共性的指令、指令的构成，以及我们课程所针对的 x86 架构的 CPU 的一些信息，比如寄存器的名称和用途等。

有这些知识，基本就能够你开始使用汇编语言了。而要真正掌握汇编语言，把它用出感觉来，只有一条途径：**在实践中使用它们**。

在这个课里，你可以用我们自己实现的编译器生成汇编代码，或者把 C/C++ 程序生成汇编代码，这样反复比较对比，你就会去除对汇编语言的陌生感。到最后，你会发现汇编语言其实真的挺简单的。

除了汇编语言这个问题外，还有一点我想跟你说一下。其实，我们要实现一门计算机语言，是绕不开编译技术的，你可能也会担心这一点，会不会让你学了编译原理再来，我现在跟你解释一下。

你是否需要懂编译原理？

我们这门课程主要是一门实践课，所以我尽量不对一些理论性的知识做过多的要求，不需要你懂编译原理。

反之，其实很多编译方面的知识点，仅仅从理论方面学习，很难学得会，反倒是动手实现，并没有想象的那么难。比如语法分析的算法，是编译原理课的难点，但在这个课程中，我会介绍当前主流计算机语言实现的方法。你自己实现的时候，那些抽象的知识点一下子变得很具体，会产生“原来就是这么回事呀”的感觉，学习门槛一下子就降低了。

另有一些知识内容，属于工程性的知识，本来在教科书上就不多见，比如如何实现一个虚拟机等，就更需要通过实践来学习了。还有很多内容超出了编译原理的课程范围，有些国外的学校，会提供一些关于运行时的课程、计算机语言设计的课程，但都比较小众。

所以说，我们这门课程，围绕实现一门语言，用到什么技术，就介绍什么技术，横跨的知识面很大，很综合。比如在运行时方面，涉及了 AST 解释器、运行字节码的虚拟机和编译成本地代码等多个机制、多个版本的实现，我很少能见到能把这些所有内容都贯穿起来的课程或书籍。

总结起来，我们这门课，更多的是“行万里路”，而不是“读万卷书”，强调在动手实践的过程中学习。而在这个过程中，一旦获得了各个知识点的直觉理解，再反过去学那些纯理论性的内容，会更加容易。

利用好课程中的示例代码

在这个课程中，我提供了用 TypeScript 和 C 语言的示例代码。在课程的文稿中，我主要写的是设计思路，但要真正理解这些设计思路和相关知识点，不是仅仅看看文稿就够的，还是要看代码、运行代码、修改代码，加上这些实践环节后，你的学习效果会更加理想。

并且，我也鼓励你用你顺手的语言重新实现。在我的其他课程里，有很多同学就用自己习惯的语言来重写课程的示例代码，用什么语言的都有，比如 Go 语言、Swift、C++、TypeScript 等。这个优秀的传统，我希望可以在这门课里继续发扬！

当然，我也会尽量把示例代码的结构理得清晰一些，多加一些注释，让你更容易看懂。如果对代码有什么疑问或建议（我自己就经常发现旧代码里的 bug），可以在评论区给我留言，或者在码云上提 issue，我会想办法去解决。

好了，上课前的准备工作就是这些。你赶紧把环境配置好，我们就开始上手学习和实践啦！相信在这门课里，你会在很多地方产生“噢，原来是这样！”的感受，帮助你打通计算机技术的奇经八脉，真的很爽！

来吧，我在课程里等你！

© 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 [开篇词 | 让我们来写一门计算机语言吧](#)

下一篇 [01 | 实现一门超简单的语言最快需要多久？](#)

精选留言 (1)

[写留言](#)

彩色的沙漠

2021-08-10

老师里面的Token字符串生成是用那个库吗？上面的代码对Token的结构不清楚

作者回复: 没太明白您的意思。

没用哪个库，都是自己手写的代码。

 1

