COOLIO：面向课堂面向对象语言的语言支持扩展

Linhan Li 乔治梅
森大学 美国

ThanhVu Nguyen
乔治梅森大学 美国

arXiv:2302.04926v1 [cs.PL] 9 Feb 2023

摘要

COOL 是一种面向对象编程语言，许多本科和研究生课程都使用它来教授编译器设计。由于大多数学生不熟悉这种语言，而且代码编辑器和集成开发环境通常缺乏对 COOL 的支持，使用 COOL 编写代码和测试程序对学生来说是一项负担，导致他们无法充分理解该语言和编译器的许多重要且高级的特性。在本文中，我们介绍了 COOLIO，这是一个旨在为流行的 VSCode 集成开发环境提供 COOL 支持的扩展。COOLIO 提供：(i) 通过词法分析和语法分析为 COOL 语言提供语法高亮；(ii) 语义感知自动补全功能，帮助学生减少代码编写量，并减轻记忆陌生 COOL 文法和语法的负担；(iii) 通过 VSCode编辑器向学生提供底层 COOL解释器/编译器的相关反馈（例如错误信息、类型信息），以辅助调试。我们相信，COOLIO 将帮助学生享受编写 COOL 程序的过程，从而学习并欣赏更高级的编译器概念。

关键词

COOL、编译器、语言支持、语法高亮、自动补全、用户反馈、集成开发环境

ACM 引用格式：

Linhan Li 和 ThanhVu Nguyen。2022年。COOLIO：课堂面向对象语言的语言支持扩展。载于*ACM*会议论文集（*Conference'17*）。ACM，纽约，美国纽约州，4页。https: //doi.org/10.1145/nnnnnnn.nnnnnnn

1 引言

COOL（课堂面向对象语言）是一种常用于本科或研究生编译器课程的语言。这些课程包括斯坦福大学、加州大学伯克利分校、内布拉斯加大学林肯分校、弗吉尼亚大学、密歇根大学和范德堡大学开设的传统课程，以及 Coursera、edX 和斯坦福在线提供的在线课程。这种语言规模足够小，学生可以在一个学期内为其实现完整的解释器或编译器，同时又足够强大，能够支持现代编程语言的主要概念，包括多态、继承、动态分派、类型检查和自动垃圾回收。正如 COOL 语言的作者 Alex Aiken 所述，COOL 是一种"面向对象、静态类型且具备自动内存管理功能"的语言 [1]。

尽管 COOL 是许多编译器和程序设计语言课程所选用的语言，但学生对 COOL 的语法和语义并不熟悉。因此，学生往往只编写简单的 COOL 程序来测试其编译器实现。此类简单测试会让学生忽略语言的许多重要特性，进而忽略编译器处理这些特性所需的关键设计（例如，动态分派、继承和复杂的内存使用）。这并不是因为学生不愿意创建更复杂、更全面的测试，而是因为在没有常用代码编辑器的帮助时，他们并不热衷于编写冗长复杂的 COOL 程序。

相关工作。目前，有一些编辑器和集成开发环境扩展可为 COOL 提供支持，例如适用于 Sublime Text 的 sublime-cool-highlighter[1] 、适用于 Atom 的 atom-language-cool[2] ，以及适用于 VSCode 的 language-cool[3] ，它们仅为 COOL 关键字提供非常基础的语法高亮。此外，它们缺少许多基本的语法高亮功能，例如，用户定义的类和变量未能得到正确的高亮或缩进。此外，尽管语法高亮很有用，但不足以促使用户使用 COOL。这些扩展不包含自动补全和错误报告等重要功能，因此学生仍需记住所有语法，或反复查阅手册，或运行提供的参考编译器来检查语法错误。事实上，由于这些扩展缺乏维护（例如，language-cool 自 2016 年以来就未更新），甚至连 COOL 的基础功能也未能妥善覆盖，大多数学生反而只是在纯文本模式下编写 COOL 程序。

本研究。为了帮助学生学习 COOL，我们开发了 COOLIO——一款功能丰富、便于移植的 COOL 扩展，可用于支持日益流行的语言服务器协议（LSP）[5]的主流集成开发环境。本文将通过 VSCode 集成开发环境展示 COOLIO。VSCode 深受学生喜爱，也是一款广受欢迎的集成开发环境（始终位列最受欢迎的集成开发环境之列）。此外，COOLIO 也适用于任何支持 LSP 的集成开发环境，例如 Atom、Eclipse、Emacs、SublimeText 和 Vim。

COOLIO 使用 COOL 词法分析器和解析器中的正则表达式来识别 COOL 语法，从而支持对整个 COOL 语言进行语法高亮。COOLIO 还提供了现代集成开发环境应有的快速实用的自动补全功能，例如代码建议和自动结构补全，所有这些功能都有助于学生编写更正确的代码，并减轻记忆 COOL 语法与文法的负担。

允许免费制作本作品全部或部分内容的数字或纸质副本，供个人或课堂使用，但不得为牟利或商业利益而制作或分发副本，且副本须包含本声明及首页上的完整引文。对于本作品中由 ACM 以外的其他方拥有版权的组成部分，必须尊重其版权。注明出处后允许摘要引用。若要以其他方式复制、再版、发布到服务器或重新分发给他人，须事先获得明确许可和/或支付费用。请通过 permissions@acm.org 申请许可。
Conference'17，*2017*年7月，美国华盛顿哥伦比亚特区© 2022年计算机协会。
ACM ISBN 978-x-xxxx-xxxx-x/YY/MM...$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

[1]https://github.com/princjef/sublime-cool-highlighter[2]
https://github.com/princjef/atom-language-cool[3]
https://marketplace.visualstudio.com/items?itemName=maoguangming.cool

```
1  class Main inherits IO{
2      main():Int{{
3          out_string("Hello, world.\n");
4          let
5              a:Int <- 0
6          in
7              {
8                  -- if a is smaller than 10, add 1
9                  (* else, break *)
10                 while
11                     a < 10
12                 loop
13                     a <- a+1
14                 pool;
15                 a;
16             };
17     }};
18 };
```

F图 1：在 VSCode 中使用 COOLIO 编写的简单 COOL 程序 COOLIO

最后，COOLIO 利用 LSP 客户端和服务器设计，通过 VSCode 编辑器向用户反馈 COOL 的类型检查和求值信息（例如，为错误代码行添加下划线，并实时说明类型检查失败的原因），使学生能够快速理解和调试 COOL 程序。

我们相信，COOLIO 能帮助在编译器或编程语言课程中学习 COOL 的学生。这包括我们的编译器课程，该课程通常平均有约 30 名本科生和研究生；也包括其他地方开设的编译器和编程语言课程，这些课程的学生人数可能更多（例如斯坦福大学、密歇根大学的课程，或 Coursera 等在线课程）。我们已经开始在教学中使用 COOLIO，也已告知教授 COOL 相关课程的同事，让他们可以推荐学生尝试此工具。我们相信，COOLIO 提供的丰富功能将有助于学生更深入地学习 COOL，并设计出更好的编译器来支持该语言的高级特性。

COOLIO 是开源软件，可从 **https://github.com/dynaroars/COOL-Language-Support** 获取。用户可以直接在 VSCode 集成开发环境或 Marketplace[4] 中轻松搜索并安装。

2 COOL

COOL 是一种面向对象语言，在设计和语法上与 Java 有一些相似之处。图 1 展示了一个简短的 COOL 程序，它会打印 HelloWorld（第 1—3 行）并执行一个简单的循环。COOL 手册称为 CoolAid [2]，，提供了该语言的形式语法和语义，下面将对此进行概述。

语法。 COOL 的语法设计得很简单。这使学生能够为词法分析器和解析器编写正则表达式（regexes），以识别语法正确的 COOL 程序。标准的 COOL 规范不包含列表/数组结构、多线程/多进程支持或异常处理等高级特性（这些功能通常作为家庭作业或额外加分作业的内容）。

表 1 展示了 COOL 的语法和上下文无关文法。COOL 程序由一组 COOL 类组成，每个类由

[4]https://marketplace.visualstudio.com/items?itemName=Linhan.cool-language- support

表 *1*：*Cool* 语法与文法 程序 *::=* [*class;*]$^+$类 *::=* 类 类型 [继承 类型] *{* [特征;]**}* 特征 *::=* 标识符*(*[形式参数[,形式参数]*]*):* 类型 *{* 表达式 *}*| 标识符*:*类型[*<-* 表达式]形式参数 *::=* 标识符*:*类型 表达式 *::=* 标识符 *<-* 表达式 | 表达式[@类型].标识符 *(*[表达式[表达式*]]| 标识符*(*[表达式 [,表达式]*]]| 如果 表达式 那么 表达式 否则 表达式 *fi*| 当 表达式 循环 表达式 *pool*| *{*[表达式;]$^+$*}* | *let* 标识符 *:* 类型 [*<-* 表达式] [, 标识符 *:* 类型 [*<-* 表达式]]* *in* 表达式| *case* 表达式 *of* [标识符*:*类型*=>*表达式]$^+$ *esac*| *new* 类型 |*isvoid* 表达式 | ~表达式 | *not* 表达式| *(*表达式*)* |表达式 *[+|-|*|/]* 表达式 | 表达式 *[<|<=]* 表达式| 标识符 |常量*(*整数|字符串|*true|false)*⊢ $e_1 : Int$ ⊢ $e_2 : Int$ ⊢ $e_1 : Int(i_1)$ ⊢ $e_2 : Int(i_2) op \in \{*, +, -, /\}$ $op \in \{*, +, -, /\}$⊢ e_1 *op* e_2 *:* *Int* $v_1 = Int(i_1\ op\ i_2)$ ⊢ e_1 *op* $e_2 : v_1$

图 2：类型检查与操作语义规则

其特性包括属性（变量）和方法。每个类都定义一种类型，因此程序员可以通过创建新类来定义新类型及其相关数据和方法（类似于 Java 等面向对象语言）。COOL 是一种表达式语言，因此大多数 COOL 结构都是表达式。表达式在 COOL 语法中占很大一部分，但总体而言相对简单，与传统语言中的表达式相似。请注意，用于声明新变量的 let 表达式与 OCaml 等函数式语言中的 let 表达式类似。

类型检查与语义。 COOL 是一种类型安全语言，因此其编译器会对输入程序进行类型检查，确保运行时不会出现类型错误。CoolAid 手册中定义的 COOL 类型规则为 COOL 表达式提供推导规则（例如，如果 x 是整数且 y 是整数，那么表达式 x + y 的结果是一个整数）。

COOL 程序的求值采用操作语义规则来描述。与用于推导 COOL 表达式类型的类型检查规则类似，这些操作语义规则用于推导 COOL 表达式的值（例如，如果 x 是 3 且 y 是 7，那么表达式 x + y 的结果是 10）。这些规则也在 CoolAid 手册中有所说明。图 2 展示了将二元运算符 *、+、−、/ 应用于表达式 $e1$ 和 $e2$ 时的类型检查与操作语义规则。

参考编译器。 修读 COOL 编译器课程的学生通常会实现一个完整的 COOL 编译器，其中包含多个主要阶段，包括词法分析、语法分析、类型检查、语义求值和 ASM 转换。为了帮助学生调试并不断推进学习，教师通常会提供一个完整的“参考”编译器，例如用 C 或 OCaml 编写的 COOL 编译器实现所编译出的可执行二进制文件。随后，学生会使用参考编译器检查其实现的每个步骤，包括

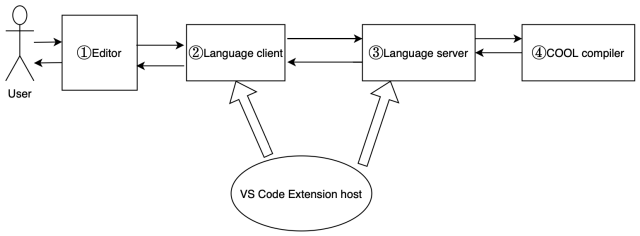


图3：COOLIO概览

输出结果和错误信息。COOLIO 依赖参考编译器向用户返回错误信息。

3 COOLIO

COOL语言服务器扩展（COOLIO）可通过VSCode的扩展窗口或第§1节所示的GitHub代码库获取。当用户打开扩展名为.cl的COOL文件时，它会自动激活，并为编辑器中打开的COOL程序添加语言支持。COOLIO由图3所示的四个组件组成。

① 该**编辑器**为用户提供图形用户界面，并支持语法高亮和自动补全等多种编辑功能。它还通过应用程序接口与语言客户端（②）交互，以显示来自COOL 编译器的错误信息。② **语言客户端**通过发送和接收信息与编辑器及语言服务器交互，例如获取已打开文件的内容、光标位置或显示错误信息。每当客户端被激活时，它都会创建一个语言服务器并向服务器发送数据。③ **语言服务器**是我们COOL 编译器的代理。当收到来自客户端的请求时，服务器会检索源代码并将其发送给COOL 编译器进行分析。分析结果会被组织成LSP响应格式，并返回给客户端以供显示。④ COOL参考**编译器**执行标准的编译阶段，例如语法分析、类型检查，并将结果（例如警告和错误信息）传达给语言服务器。

3.1 语法高亮

COOLIO 中的语法高亮由 VSCode编辑器负责处理。编辑器使用 TextMate 配置文件中指定的正则表达式规则对源代码进行词法分析。随后，编辑器会根据词法单元定义的角色以及集成开发环境当前主题指定的颜色，为集成开发环境中显示的词法单元着色（例如，在某些主题中，注释显示为蓝色，而在其他主题中则显示为红色）。

对于 COOLIO，我们创建了一个 TextMate[3] JSON文件，其中包含 COOL 所有关键字、词法单元等对应的正则表达式。图 4 展示了一个 TextMate 配置片段，其中包含用于匹配行注释和类标识符的正则表达式。表 2 展示了 COOL 中若干主要组件的语法高亮正则表达式。

用户打开 COOL (.cl) 文件时，COOLIO 扩展便会激活，并使用这些规则进行语法高亮。虽然正则表达式可能很复杂，但 COOL 参考编译器已经

```
1  "scopeName": "source.cool",
2  "name": "COOL",
3  "文件类型": ["cl"]
4  "patterns":[
5    "line_注释": {
6      "begin": "--", "end": "$\\n?",
7      "name": "comment.line.double-dash.cool"
8    },类": {
9
10     "match": "[A-Z][a-z]*",
11     "名称": "entity.name.type.类.cool"
12   }]
```

图 4：COOLIO 中的语法高亮配置

表 2：语法高亮规则

COOL 构造	正则表达式描述
整数	不带前导零的非负整数
ID（类）	以大写字母开头的字母数字字符串
ID（其他）	以小写字母开头的字母数字字符串
特殊 ID	self、SELF_类型
字符串	由双引号括起来的字母数字字符串
注释	以“–”开头的行，以及由“(*”和“*)”括起来的块
关键字	*类、else、false、fi、if、in、inherits、isvoid、let、loop、pool、then、while、case、esac、new、of、not、true*

包含用于词法分析和语法分析的类似正则表达式。因此，我们对这些正则表达式进行调整并重复使用。

示例。图 1 展示了 COOLIO 中的语法高亮。在默认的 *VSCode Light* 主题中，关键字为黑色，控制流结构为紫色，类名为青色，整数常量为绿色，等等。

3.2 自动补全

COOLIO 支持标准的自动补全（在 VSCode 中也称为“IntelliSense”[4]）。用户只需输入几个关键字，集成开发环境就能建议并自动补全常见的代码结构。例如，当 COOLIO 检测到用户正在输入关键字 if 时，它会询问用户是否要将其替换为 COOL 代码片段“if condition then expression else expression fi”。这种自动补全简单直接，却有助于在集成开发环境中编写代码，减少开发者的输入工作量和语法错误。

为提供语法和结构均正确的代码片段，COOLIO 的自动补全功能了解 COOL 的语法和文法。与语法高亮（§3.1）类似，这通过编辑器层面实现，使用用户提供的 TextMate 配置文件。该 TextMate 文件由一些规则组成，用于将前缀字符串（例如 let）映射到代码片段（例如 let var ... in ..）。代码片段可以包含多个用于任意子表达式的占位符，用户可以使用“Tab”键在它们之间切换。扩展加载配置文件后，编辑器会自动尝试将前缀与用户输入的字符串进行匹配。如果输入与任何代码片段的前缀匹配（可以是完全匹配，也可以是部分匹配），编辑器就会在下拉式手册中列出该代码片段的名称和说明，供用户选择。

```
"COOL_类_继承": {
    "前缀": "类",
    "主体": [
        "类${1:名称} 继承 ${2:对象}{",
        "\t${0:bodv}".
        "}:"].
    "description":   "COOL: 类 inherits"}
```

图 5：TextMate 自动补全配置

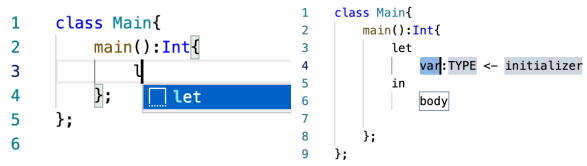


图 6：COOLIO 中的自动补全

图 5 展示了 COOLIO 的 TextMate 配置示例。其中，“body”逐行定义代码片段，而“${...}”则标记占位符。该代码片段定义了 COOL 中类定义的代码片段（即用户输入 class 时，COOLIO 可以填入用于定义 COOL 类的框架）。

示例。图 6 展示了一个示例：当 COOLIO 察觉用户想通过 let 关键字声明一个新变量时，它会生成新变量声明代码片段，并提供用于指定类型和初始值的相应占位符。

3.3 与用户的交互

COOLIO 使用 LSP 进行反馈通信，即将后端 COOL 编译器提供的信息（例如错误和警告）显示在前端编辑器中。LSP 的设计允许在独立进程中进行分析，并通过进程间通信将结果传递给编辑器（因此，在服务器中进行的分析不会影响用户与编辑器的交互）。

COOLIO 包含一对语言客户端和语言服务器。语言服务器充当代理，与 COOL 参考编译器通信。当用户保存代码更改时，服务器会对代码调用 COOL 编译器，并将输出发送给客户端。语言客户端充当编辑器与语言服务器之间的中间层。它接收来自服务器的数据（例如错误信息），并调用编辑器的应用程序接口将其显示给用户。

对于 COOLIO，要将来自 COOL 编译器的错误信息等数据报告给客户端，需要对错误信息进行格式化和翻译，使其转换为编辑器能够理解并显示的形式。具体来说，当编译器报告程序中存在错误或警告时，编辑器会显示行号、编译阶段以及详细的错误信息。这些错误和警告可能出现在任何编译阶段，例如无法解析某些表达式，或类型关联不正确。

请注意，COOL 编译器使用行号记录错误位置，而语言服务器协议要求以字符位置表示错误位置。这个问题由语言服务器处理：我们用相应换行符的位置替换了行号。

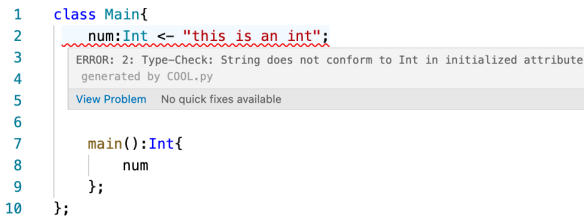


图 7：显示错误信息

示例。 图 7 展示了 COOLIO 如何通过编辑器向用户传达错误信息。在第 2 行，用户将一个字符串常量赋给类型为整数的变量“num”，这是不允许的（因为类型不兼容，而且 COOL 不支持类型转换）。后端编译器捕获了此错误并将其报告给用户。红色波浪线标出了错误位置；如果用户将鼠标悬停在线上，就会显示详细的错误信息。

4 结论

我们介绍了 COOLIO，这是一种语言服务器协议（LSP），可扩展现有的集成开发环境和编辑器，使其支持 COOL 编程语言。与 C++ 和 Python 等其他语言的 LSP 类似，COOLIO 旨在通过完整的语法高亮、IntelliSense 代码自动补全，以及后端编译器与前端编辑器之间丰富的错误和通信机制，让 COOL 代码编写更加轻松。

COOLIO 的一个间接目标，是向研究人员展示一种简单的方法，让他们的研究成果对熟悉集成开发环境、但不熟悉研究工具的用户而言更加可见且易于使用，因为研究工具通常以命令行工具的形式编写。研究人员可以将其研究工具（例如静态或动态分析工具）集成到 LSP 中，并以 COOLIO 与 COOL 编译器交互的相同方式与用户交互。例如，开发者除了创建命令行故障定位或程序修复工具外，还可以创建一个将该工具与集成开发环境连接起来的 LSP，让用户通过任何支持 LSP 的代码编辑器直接与工具交互以修复代码。考虑到 VSCode 等集成开发环境的流行及其强大的扩展生态系统，我们相信这能让研究工具对用户更具吸引力；用户通常乐于尝试新的扩展，而这些扩展就像 VSCode 中的扩展一样，易于安装和使用。

参考文献

[1] Alexander Aiken。1996年。《Cool：用于教学编译器构造的可移植项目》。《*SIGPLAN* 通讯》31卷7期（1996年7月），19–24页。[2] Alexander Aiken。 1996 [在线]。COOL。http://theory.stanford.edu/~aiken/ software/cool/cool.html，访问于2023年2月13日。[3] MacroMates。2021年。TextMate：适用于 macOS 的文本编辑器。https://macromates.com/，访问于2023年2月13日。[4] 微软。2021年。Visual Studio Code 中的 IntelliSense。https://code.visualstudio. com/docs/editor/intellisense，访问于2023年2月13日。[5] 微软。2022年。语言服务器协议官方页面。ttps://microsoft. github.io/language-server-protocol，访问于2023年2月13日。