**天津工业大学**

本 科 生 毕 业 设 计（论 文）

面向中学生的“竞赛算法”可视化案例库的设计与实现

**学 号：**　1711640118

**姓 名：**　叶文滔

**专 业：**　计算机科学与技术

**学 院：**　计算机科学与技术学院

**指导教师：**　王作为

**职 称：**　讲师（高校）

**完成日期：**　2021年5月26日

# 摘　　要

随着信息技术的发展，计算机正被越来越多人所触及并应用。计算机作为第四次工业革命（也称工业4.0）的核心工具，其在一个国家内的普及程度将决定了第四次工业革命的广度，因此国家推出了各种各样的政策，引导计算机在青少年中的普及。其中便包括了举办计算机程序设计竞赛、对中小学校建立并推广计算机学科提供的政策及资金扶持等。这也使得越来越多的学生对计算机产生兴趣，并参与到计算机编程的学习中。

对于青少年的编程学习，一般可分为“编程思维”、“数据结构”、“算法”三个阶段。“编程思维”着重学习C语言程序的运行原理，如顺序、分支、循环三大结构；“数据结构”着重学习数据在计算机中的存放方式，使学生了解数据类型在计算机程序中的意义；“算法”则是学习使用计算机编程解决生活中的数学问题的方式。本项目主要以中学青少年计算机爱好者为目标人群，以“竞赛算法”所涉及的核心算法为教学目的，利用计算机相对成熟的技术来实现相关算法的可视化教学。并引入不同的人机交互，让学生更加直观地理解竞赛相关算法，提高学生的学习兴趣。

本课题实现了一个可即时编程并实时演示数据结构的可交互编程系统，引入面向中学生“竞赛算法”的教案，实现对教案中相关算法的可视化演示。该系统使用编译原理技术实现对C语言程序的识别和数据提取，并采用web技术实现过程数据可视化。这使得该系统可运行于各种系统与形态中，方便教师和学生的使用。该系统既可以加深学生对基本算法的学习和理解，又能够实现动态演示不同参数下的各种算法的状态及变化，实现中间的数据的输出，提高学生的动手及实践能力。

关键词：中学生；计算机；编程；C语言；竞赛算法；可视化；Web开发

# **Abstract**

With the development of information technology, computers are known and being widely used by more and more people. As the core tool of the fourth industrial revolution (also known as Industry 4.0), the popularity of computers in a country will determine the breath of the fourth industrial revolution. Therefore, our country has launched a variety of policies to guide the popularity of computers among teenagers. These include holding computer programming competitions, providing policy and financial support for the establishment and promotion of computer science in primary and middle schools. This also makes more and more students become interested in computers and participate in the learning of computer programming.

Generally, learning process of computer programming for teenagers can be divided into three stages: logic, data structure and algorithm. “Logic” focuses on learning the operating principle of C language programs, such as the three structures of sequence, branch, and loop. “Data structure” focuses on learning how data is stored in the computer. “Algorithm” focus on learning how to solve mathematical problems in life with computer programming. This project is mainly aimed at young computer enthusiasts in middle schools, with the core algorithm in “competition algorithm” as the teaching purpose, and use some relatively mature technology to achieve the visual teaching of those algorithms. It also introduces different human-computer interactions to allow students to understand competition algorithms more intuitively and increase their learning interest.

This topic implements an interactive programming system that can be programmed instantly and show data structure in real time. With “competition algorithms” for middle school students teaching plans introduced, it can realize the visual demonstration of related algorithms in the teaching plans. The system uses compilation principle to recognize and extract data from C language programs, and uses web platform to realize process and data visualization. This allows the system to run in various systems and forms, makes it convenient for teachers and students to use. The system can not only deepen student’s learning and understanding of basic algorithms, but also can dynamically demonstrate the state and changes of various algorithms under different parameters, and the middle-state data output, which can improve student’s hands-on and practical ability.

**Keywords**: Middle school students; computer; programming; C language; competition algorithm; visualization; web;

# 目　　录

[第一章 绪论 1](#_Toc72603840)

[一、课题研究背景 1](#_Toc72603841)

[二、国内研究现状 1](#_Toc72603842)

[三、课题研究内容 2](#_Toc72603843)

[四、本文主要结构 3](#_Toc72603844)

[第二章 相关开发技术介绍 4](#_Toc72603845)

[一、TypeScript 4](#_Toc72603846)

[二、React 5](#_Toc72603847)

[三、webpack 6](#_Toc72603848)

[四、node.js 7](#_Toc72603849)

[第三章 系统总体设计 9](#_Toc72603850)

[一、需求分析 9](#_Toc72603851)

[二、系统功能设计 9](#_Toc72603852)

[三、系统整体结构设计 11](#_Toc72603853)

[第四章 基本模块详细设计方案 14](#_Toc72603854)

[一、数据结构 14](#_Toc72603855)

[二、代码服务 16](#_Toc72603856)

[三、页面视图 18](#_Toc72603857)

[(一)菜单 18](#_Toc72603858)

[(二)图形演示区 19](#_Toc72603859)

[(三)代码编辑器 19](#_Toc72603860)

[四、教案 20](#_Toc72603861)

[五、教案加载器 20](#_Toc72603862)

[第五章 编译器模块详细设计 22](#_Toc72603863)

[一、编译原理概述 22](#_Toc72603864)

[二、编译器的结构 22](#_Toc72603865)

[三、词法分析 24](#_Toc72603866)

[四、语法分析 27](#_Toc72603867)

[五、变量表分析 29](#_Toc72603868)

[六、解释执行器 30](#_Toc72603869)

[第六章 课程设计 31](#_Toc72603870)

[一、 桶排序 31](#_Toc72603871)

[二、插入排序 32](#_Toc72603872)

[三、冒泡排序 33](#_Toc72603873)

[第七章 总结和展望 34](#_Toc72603874)

[参考文献 36](#_Toc72603875)

[附　　录 37](#_Toc72603876)

[一、相关代码 37](#_Toc72603877)

[(一) TokenType枚举值（对应词法分析自动机状态值） 37](#_Toc72603878)

[(二) C语言简化版文法 39](#_Toc72603879)

[二、相关资料中英文对照 41](#_Toc72603880)

[(一) 英文原文 41](#_Toc72603881)

[(一) 中文翻译 43](#_Toc72603882)

[致　　谢 45](#_Toc72603883)

# 第一章 绪论

## 一、课题研究背景

C语言作为计算机领域的一门经典语言，在过去的十几年间被持续广泛地使用，也被国内各大高校和各类计算机培训机构作为首选编程语言。其实践性高、综合性强，相对于其他高级语言更贴近计算机的运行原理，大部分语法亦可应用到其他语言中去，因此相当多人选用此语言作为计算机编程基础知识学习以及系统级的程序编写工具。这也使得C语言得以在Tiobe编程语言排行榜中长期位列第一或第二[1]。（在过去的20年间亦有很多年由Java承担榜首的位置，这与网站后端开发与移动端开发有较大关系。）

正因如此，国内对青少年的计算机编程教学，目前大多数使用的是C语言。尽管C语言相对其他大多数高级语言具有更低的功能性和更高的学习难度，但由于其相对贴近计算机的运行原理，语法与其他大多数语言相近，因此可作为青少年学习编程的首选语言。

但随着计算机行业的发展，国内计算机教育的问题也逐渐显现了出来——计算机学科由于其本质的枯燥性，使得多数学生无法独立进入此领域。尽管一些民营计算机教育中心拥有较良好的教育方式能使学生快速上手，但在传统学校中，受到国内“成绩决定未来”的恶性气氛影响，学校配备的计算机相关师资力量仍不足以使学生对计算机学科具有较好的理解。因此改进计算机学科的教学方法，是推进计算机教育的重要一环。

## 二、国内研究现状

编程语言的学习一般分为被动式学习和主动自学两种。相对于其他语言，国内C语言学习者大多数情况下的学习方式是被动学习，即接受高等教育学校提供的相关教学。这种教学方式往往更注重基础能力的积累，不注重新功能的传授，因此学习速度并不高。但由于存在任务性需求，加之“教师为主体、学生被动参与”的传统学习方法已沿用多年，因此绝大多数人都可以接受。

对于接受编程思想能力更强的人来说，脱离传统教学方式的自主学习往往是更优的方案。由于C语言的火爆性，目前能给C语言初学者提供的自学书籍教材相当多，如C语言之父合著编写的书籍——《C程序设计语言》，全面地讲述了C语言的基础语法和见解；《C语言程序设计：现代方法》，以一种独特的视角介绍C语言及其变化；面向C语言专业研究者的《C Primer Plus》，给读者更好地讲述C语言的深层用法及原理；数据结构介绍书籍《大话数据结构》，以通俗有趣的方式介绍数据结构的概念；算法类书籍《啊哈！算法》，从实际应用出发，以幽默的语言和插图来讲解算法；《算法导论》，深入浅出、全面地介绍了计算机算法。

相比书籍，现在更为流行的自学方法是通过编程自学网站平台进行学习。这些网站有以教学为主，在线编辑器为辅的菜鸟教程（ruboob.com）、w3school（w3school.com.cn）；以在线编辑器动手操作优先的ITbegin（itbegin.com）、以真题练习为主的力扣（leetcode-cn.com）、牛客网（newcoder.com）等。它们能提供除C语言外更多的编程语言及框架的教案，扩展性和时效性都更好。人们可以根据自己的喜好和学习环境，选择自己喜欢的方式、喜欢的教材进行学习。

同时，也有通过视频教学进行自主学习的方案可选。视频教学方案结合了传统教学和其他几种自主学习方式的优点，在讲授式教学的同时可以随时调整速度和进度，可以适应更多场景，解决了上课可能不太方便带电脑进行当场实践的问题，是一种不错的选择。相关网站有慕课网（imooc.com）、中国大学MOOC（icourse163.org）等。

## 三、课题研究内容

本课题针对现有的C语言在线编程环境无法为用户提供调试功能、IDE无法提供图文教学功能、教案书籍无法提供与程序运行实时对应的图文演示功能进行了功能整合，借鉴了国内C语言学习环境的研究成果，针对学生和教室对C语言的学习和教学需求，设计了一套能在网页和桌面客户端内完成程序编写、程序解释单步执行、断点调试、实时图形展示的系统。

本课题重点对如何实现C语言解释编译器的关键技术做了研究，通过此套系统提供程序运行的变量信息和程序运行状态，为上层服务；并在此基础上使用Web技术实现页面框架的搭建，最终可使用这套平台为教案和程序搭起一座桥梁，进而完成本系统的功能实现。

## 四、本文主要结构

本文内容分为以下几个章节：

第一章为绪论。首先对本课题的研究背景和国内的研究现状做了简要介绍，然后说明本课题的主要研究内容及内容安排。

第二章为相关开发技术介绍。简要介绍了本工程应用到的几项重点技术，包括工程语言 TypeScript、前端开发框架React、工程打包处理框架webpack，以及基于V8引擎的JavaScript运行时系统node.js。

第三章为系统总体设计。首先根据课题进行了需求分析，并根据需求设计出相应的软件系统解决方案。然后从两个视角描述了系统的整体结构设计，并简要介绍了系统运行的流程。

第四章为基本模块设计。分别介绍了核心数据类型以及代码服务、页面视图、教案、教案加载器四个模块的设计。

第五章为核心模块设计——编译器。编译器用于对C语言程序进行编译操作，以为代码编辑器提供语法高亮、变量表、指令队列等功能。它是本系统的低层核心功能，包含了词法分析器和语法分析器。

第六章为课程设计。简要地列举了本系统已实现的一些算法案例库，并给出了程序的实际运行效果。

# 第二章 相关开发技术介绍

本章主要介绍了本系统实现过程中使用到的相关开发技术，包括工程语言 TypeScript、前端开发框架React、工程打包处理框架webpack，以及基于V8引擎的JavaScript运行时系统node.js。

## 一、TypeScript

TypeScript是JavaScript的超集，它可以编译成纯JavaScript，是一款可以在几乎任何浏览器、任何计算机和任何操作系统上运行的开源编程语言[2]。

在基于浏览器的编程语言发展历程中，JavaScript于1998年由Netscape（网景）推出后大获成功[3]，并于后期浏览器开发语言的竞争中取得优势地位，由ECMA国际标准化组织统一了ECMAScript标准[4]，并占据了所有浏览器编程语言的市场。简单来说，目前在主流浏览器上支持的语言是JavaScript，它基于ECMAScript标准持续迭代。

JavaScript作为一款专用于浏览器环境的脚本语言，对小型项目支持良好，对新接触程序开发的程序员表现友好，因此它能在网络大普及的年代迅速地占领市场。它支持面向对象程序设计、命令式编程，以及函数式编程。它提供语法来操控文本、数组、日期以及正则表达式等[5]。但其动态类型、基于原型等设计方式则在大型项目中存在明显的短板。动态类型使得开发人员在开发过程中无法获得来自编辑器和编译器的提示，发生错误的时机被推迟到了生产环境，容易造成不可预知的后果，只能通过精确的文档去尽量避免这个问题。加之原型链式设计方法并不是面向对象编程的良好方法，因此非常有必要引入一些更优的方案。

TypeScript便是一款基于JavaScript的编程语言。它通过在JavaScript的基础上添加静态类型定义构建而成。类型提供了一种描述对象形状的方法，可以帮助提供更好的文档和代码提示，可以让TypeScript在开发过程中就验证代码可以正常工作，对于大型工程的代码可读性和可维护性起到强大的作用。而它的类型便不是强制的，而是渐进式的。这允许在编写TypeScript的时候忽略类型推导，并在后续的开发过程中再进行定义与引用，或者直接使用类型推断，即可使用IDE快速地完成类型处理的工作，无需编写额外的代码即可获得大量功能。

TypeScript拥有活跃的社区。很多JavaScript的第三方库都提供TypeScript的类型定义文件。我们可以通过一些数据来了解TypeScript的流行趋势。



图2.1 JavaScript和TypeScript百度搜索指数对比（近10年）

## 二、React

React（也可称React.js或ReactJS），是一个用于构建用户界面的 JavaScript 库[6]。

React具有如下特点：

1. 声明式。相对于命令式编程方法，此编程方法能更轻松地更新组件的状态，当数据改变时能有效地更新并渲染组件。这使代码更加可靠，方便调试。

2. 组件化。可通过React创建拥有各自状态的组件，再由这些组件构成更加复杂的UI。React组件通过使用JavaScript或JSX编写，相比于Vue中使用的模板式写法，可以更轻松高效地控制数据与组件的关联关系与渲染逻辑，并使得状态与DOM分离。

3. 多平台。React不仅可用于网页构建，还可使用Node.js进行服务器渲染，或者使用React Native开发原生移动应用。这使React可实现“一次学习，随处编写”。

React使用了一种新的文件格式“JSX”。这是一种将JavaScript与HTML结合起来构建React组件（后续Vue也通过插件支持了这种格式）的语法。它允许开发者将HTML标签直接作为JavaScript对象使用，亦可在HTML标签中直接穿插JavaScript逻辑，其逻辑类似于Servlet。使用JSX编写的代码经过Babel编译后即可转换成JavaScript代码[7]。这使得开发人员在控制渲染逻辑上拥有了极大的便利性。

React由Facebook创建。起初仅在Facebook内部使用，后来此框架由于太受欢迎，因此于2013年开源。此后快速占据前端主流框架的榜首。其流行指数如图2.2所示。

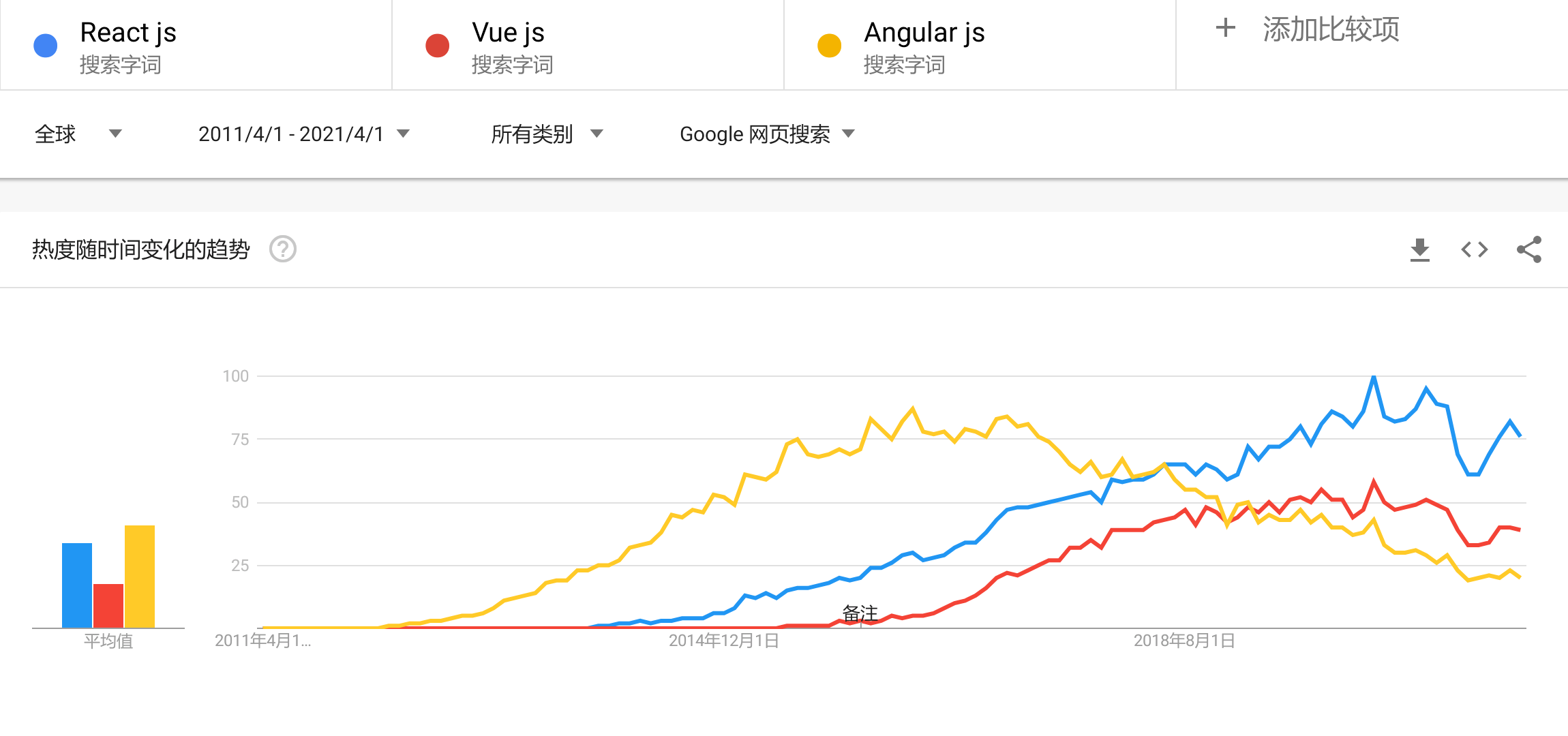


图2.2 ReactJS、VueJS、AngularJS 谷歌搜索指数对比（近10年）

## 三、webpack

webpack是当前前端最热门的前端资源模块化管理和打包工具[8]。

传统的前端脚本引入方式是在DOM中新增script标签，并将其src属性设置为需要引入的JavaScript文件地址。在实际开发中，我们往往会依据模块化的思想，将单个JavaScript文件作为一个模块。而模块与模块之间往往具有较强的依赖性，即模块之间会形成一套依赖链网络。依靠开发者自行计算并导入费时费力，而且每需要一个JavaScript文件便要多进行一次HTTP请求，对性能有影响，因此最佳的方式便是将所有文件都打包起来，一次引入。webpack 的工作便是对JavaScript工程的文件进行打包工作的库。其工作能力如图2.3所示。

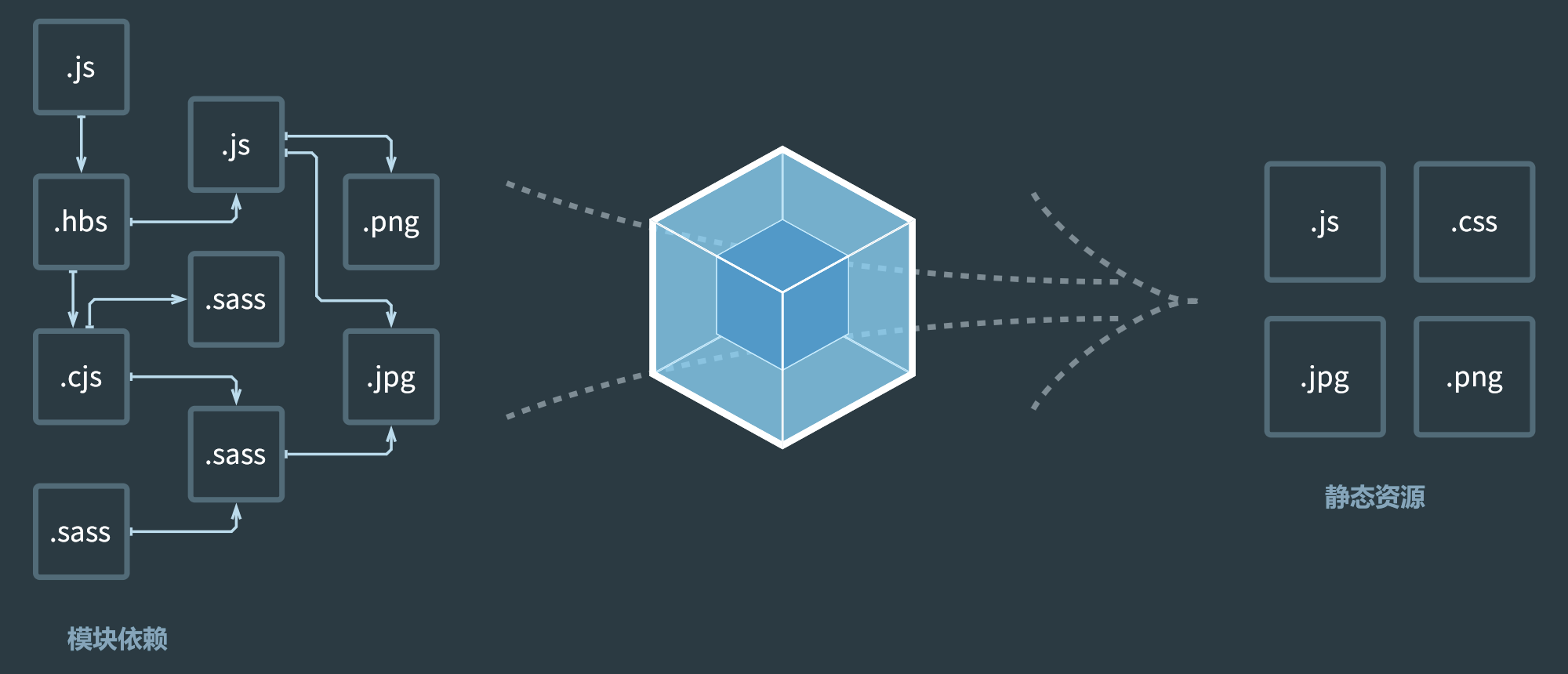


图2.3 webpack能力示意图

webpack可与其他前端框架高度集成，支持单页面和多页面工程的打包，支持自定义文件格式（或模块）的引入，支持按文件格式分发loader（如TypeScript、Babel、Sass、Stylus等）进行引入，支持插件对代码进行各种后处理，支持生成map文件以供调试，支持热更新，以及更多此处未提及的功能。

这些功能使得一个庞大的工程在浏览器引入之前便被浓缩成少量几个文件，开发人员无需关注模块之间的具体导入顺序，可以如同其他编译型语言一样自由地编写模块；在需要进行调试时，热重载和map文件生成功能可使得本身仅支持JavaScript的开发者工具能支持任意语言的调试，并可在代码改动后在页面内直接生效，无需重载页面。这对于工程开发是极为有利的功能。

webpack现已被众多JavaScript框架选为首选打包工具。如VueJS的vue-cli、ReactJS的create-react-app等[9,10]需知道JavaScript及框架打包的内部细节，直接使用此工具即可快速地生成对应框架的应用。

## 四、node.js

Node.js 是一个基于 Chrome V8 引擎的 JavaScript 运行时[11]。

Chromium中使用的V8引擎V8（谷歌提供的JavaScript引擎）广受大众好评，node.js使V8引擎可以脱离于浏览器运行，实现更多依赖本地的功能，如Web服务器，以及本文需要使用到的本地代码编译。

JavaScript通常被认为是一门解释型的语言，但是现代的JavaScript引擎不再只是解释JavaScript，也会对其进行编译。自2004年Google地图的引入以来，JavaScript已经从一门通常执行几十行代码的语言演变为能在浏览器中运行具有成千上万行代码的完整的应用程序。在这个新世界中，编译JavaScript非常有意义，因为尽管可能需要多花费一点时间来为JavaScript做准备，但是一旦完成，则它会比纯解释型的代码具有更高的性能。上文中提到的TypeScript、React、webpack便是在前端工程工具链中的重要成员，他们相互合作，将开发者的代码在node.js平台上进行编译。可以认为，这是一套使用JavaScript将其他语言编译成JavaScript的平台。

Node.js自带一个包管理器——npm，是一个下载和管理Node.js依赖包的软件，也是一个存放公共包的平台（也可以用于公司的私有软件包访问）[12]。使用npm，可以便捷地连接到Node.js的开源世界，分享和使用成千上万的开源代码。现在npm已成为世界上最大的包管理分发平台，如图2-6所示，其包存有量已远超排名第二的Maven Central。通过module count网站上抓取的数据，我们可以将各大公共包托管平台近十年的包数量做成一个图。由图2.4可见，npm的包数量在2021年已经远超排名第二的Maven Central (Java) 数倍。

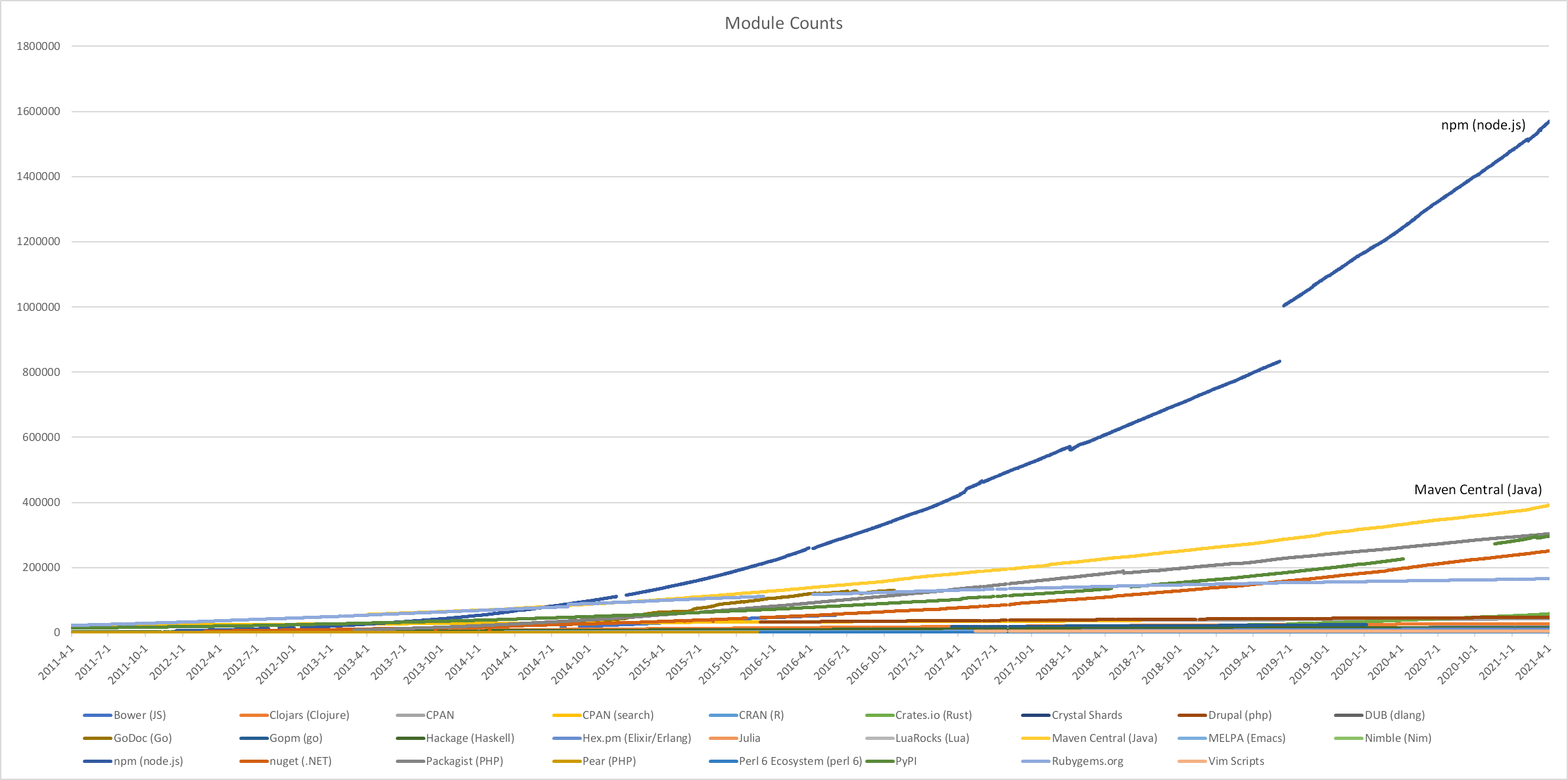


图2.4 module counts统计的包托管平台包数量变化趋势图（近十年）

本工程使用yarn作为包管理工具。yarn同样使用npm仓库，但yarn拥有比npm更好的性能，特别适合作为npm的代替品在大陆地区内使用。

# 第三章 系统总体设计

本章首先对课题进行需求分析，通过分析结果对系统提供的主要功能进行了描述，然后从两个角度介绍该系统的整体结构设计，并简要介绍了系统整体运作流程。

## 一、需求分析

算法是编程中重要的一环，而实现优秀的算法往往需要有扎实的编程思想基础。事实上，对于编程思想的教学，在早些时候就已经提前到高中开始进行了。使用过人教版《数学》选修1-2的同学可能已经学过书中“流程图”的相关内容。这里的“流程图”，便是编程思想的基础。流程图编写的思想与其他数学问题的解题思想有明显不同：一般的数学问题需要在题内进行相关知识的发散性、双向性思考，从而得出结论；而流程图的构建则是线性的，构建的是过程，而不是结果。如 x=x+1，这个语句在解方程题中可以得出 x=0.5 的结论，而在流程图或编程中，这个语句仅代表将 x 的值更改为 x+1 的操作，即本质上是构建“操作”，而不是直接从语句解算结果。为了方便解释这样的事情，往往需要在草稿纸上记住程序运行的位置、当前的变量状态等信息。

尽管现代的IDE已经具备相当方便的调试功能，可以在程序运行的过程中方便地知道程序的运行位置、调用栈、内存，但这些IDE往往界面复杂，充斥着各种对新人不友好的名词，C语言初学者往往只能靠反复执行printf函数以及笔记画图来获知程序的运行状态，而C语言教学者往往也只能采用类似的方式进行教学。这种操作不仅存在明显的割裂感，也浪费了较多时间进行输出处理。特别是当学习具有具体意义的算法时，依靠IDE的调试功能进行讲解和理解确实会相当枯燥，对于初学者来说实在是一道门槛。

因此，大家希望可以拥有一套系统，能把C语言编程与算法图解联系起来，达到实时演示的目的，方便大家的学习。

## 二、系统功能设计

根据上述需求分析，可确定本系统的以下两个核心功能：源码即时编辑运行功能和算法数据演示功能。其中源码即时编辑运行功能类似于传统IDE提供的代码编辑器及调试功能，支持在网页中编辑源码并单步执行。算法数据演示功能即从代码中提取出变量表等运行状态，并即时反馈到图形演示区中。除此以外还有菜单区，可在此选择进入不同的教案。为方便用户使用，程序主界面将按从左到右的顺序分为三个功能块：菜单区、图形演示区、代码编辑器。如图3.1所示。



图3.1 整体界面示意图

下面对各个功能区进行具体描述：

1. 菜单

菜单是进入程序后的第一个主界面，此处采用ExpandableListView的设计方案，将按不同的算法功能类型分章，按相同算法功能类型的不同算法细分节，给用户提供课程。

2. 图形演示区

图形演示区将按照不同的算法实现不同的演示方法。这部分将按一个特定的规则制成不同的演示器，并在选择课程后导入。因此这部分可按实际情况设计成任何样式，并没有特定的界面方案。但不同组件的输入和输出均遵守同一套约定的规则。

3. 代码编辑器

代码编辑器是让用户输入程序代码的地方，教案会根据需求向用户预先提供示例代码，用户也可在此处按教案提示的规则输入或修改C语言代码，代码编辑器也将提供相应的语法高亮功能以辅助用户进行代码编写。此代码编辑器同时具备调试功能，即允许直接运行、单步操作、重新开始等调试器常用操作，并将代码的运行结果实时反馈到图形演示区中，使用户可以直观地看到抽象的代码在演示器中对应的具体的形象。

## 三、系统整体结构设计

该系统整体被设计为单个React工程，并且不设计对外调用其他库的功能，因此所有操作均在这个工程中进行。图3.2示意了该系统的模块结构和数据流。我们也可通过工作特征和系统功能两个视角来理解系统的模块分工。



图3.2 系统整体设计及流程图

从工作特征视角看，本系统可分为前台和后台部分。

1. 前台使用React框架构建，主要负责对用户可见的部分进行渲染，并主控、桥接系统的各个组件及其对应接口。

2. 后台部分主要负责课件的加载、代码的识别和解释执行，并将相应的处理结果返回到前台供展示。

图3-2示意图中被深天蓝色包裹的部分即为前台部分，其余为后台部分。

从系统功能角度看，本系统可分为教案、教案加载器、页面视图、代码服务、编译器五大模块。

1. 教案（灰白色渐变块）被设计成了独立的React模块，可根据不同的教学需求制作不同的教案；每个教案模块向外提供初始代码、预期表量表，向内要求提供程序的实际变量表，也可以依据实际需求添加不同的需求信息，如代码正在运行的行数等，然后通过这些信息渲染图形界面。

2. 教案加载器（淡橙色块）负责提供教案菜单，以及菜单项中对应的教案标题和教案模块。

3. 页面视图（淡蓝色块）负责渲染控制程序的主体功能模块，包括菜单、图形演示区、代码编辑器三个部分。可以认为程序的中心是页面视图主控，而其它模块则为外挂服务。

4. 代码服务（淡紫色块）负责为代码编辑器和编译器提供中间支持，包括将字符串类型的代码转换成相应的内部数据类型、按指定位置或指定方向读取代码的光标服务、根据指令队列执行代码、存储变量表和运行时内存信息等。

5. 编译器（灰蓝色块）负责对C语言程序进行编译操作。它是一个非完整C语言编译器，仅提供C语言的部分核心功能，最终产出的结果仅供该平台进行执行，不输出可执行二进制文件。它包含了词法分析器和语法分析器。通过代码服务提供的功能扫描用户代码，在词法分析器中生成词法单元（token）序列，词法单元序列通过语法分析器生成语法树，并在此过程中进行语法制导翻译和中间代码生成，产出指令队列。这些信息不是互相独立的，而是包含了相关的引用，比如代码中的每个字符在经过词法分析后都会存有到对应词法单元的引用。因此编译器与代码服务息息相关。

图3-2上的数字示意了系统的主要数据流图。从选择课件到代码编写及程序执行的主要数据流如下：

1. 程序载入后，菜单组件自动通过主控向教案加载器请求菜单。

2. 用户选择菜单上的教案后，菜单组件自动加载对应的教案组件。

3. 教案组件自带的示例代码将通过主控传给代码编辑器，其自身则作为一个React组件嵌入到图形演示区中。

4. 代码编辑器在接收到来自主控或者用户发来的代码编辑请求后，将对应动作转发到代码服务中。

5. 代码更新后，对应代码将交给词法分析器进行词法分析处理。这一步将通过词法分析自动机产出代码对应的词法单元序列。

6. 词法单元序列将进一步被送往语法分析器。这一步将通过运用C语言对应CFG的LR分析器产出语法树，然后产出变量表，用于在解释执行的过程中作为辅助。

7. 上一步最终生成的内容将将被交到代码服务中去。

8. 代码服务中的变量表将被送往图形演示区中的教案。指令队列将在运行程序时被使用。这些指令将在运行时对变量表中的变量产生影响，并最终反馈到图形中。

# 第四章 基本模块详细设计方案

在第三章中，我们把该系统按系统功能角度分成了五个模块——教案、教案加载器、页面视图、代码服务、编译器。其中，编译器为低层核心，在本系统中占据重要成分，因此将其独立为第五章。本章将先介绍本工程的一些基本数据类型，然后分别介绍代码服务、页面视图、教案、教案加载器的设计方案。

## 一、数据结构

该系统的主要工作，简单来讲可以阐述为：用户在代码编辑器中输入代码，经过一系列转换后，将相应的数据输出到视图中。而托管用户代码的模块为代码服务。因此我们从代码服务出发，设计其对应的数据结构。

代码在输入到代码编辑器后，代码服务会将这些代码按行分割，然后按字符分割，进行存放。代码服务中维护一个CodeLine类型的数据，用于存放所有代码行，其数据结构如下所示：

|  |
| --- |
| export type CodeLine = Array<CodeChar>; |

CodeChar用于存放代码行中的每个字符，其数据结构如下所示：

|  |
| --- |
| export interface CodeChar {      char: string;   // 字符      token: Token;   // 向上对应的词 token  } |

其中，“token”即词法单元，每个词法单元可以由多个字符组成，因此CodeChar中的Token存放的是词法单元的引用。

在进行代码定位操作时，通过行列即可访问到代码中的相应字符，系统也具有查找上一个字符、下一个字符等能力。在进行这些操作时，比较方便的方法是新增一种类型CodeCharWrapper，用于在CodeChar基础上夹带位置信息。其数据结构如下所示：

|  |
| --- |
| export interface CodeCharWrapper {      ln: number;     // 行      col: number;    // 列      code: CodeChar; // 代码字符  } |

同时也有它的简化版本CodePosition，用于不需要关心代码字符本身，只需要关心代码位置的步骤。其数据结构如下所示：

|  |
| --- |
| export interface CodePosition {      ln: number;      col: number;  } |

在刚才的描述过程中，提到了词法单元token。这是在编译过程中进行词法分析时的一个概念。每个词法单元对应若干个字符、具有一个状态。Token的数据结构如下所示：

|  |
| --- |
| export interface Token {      type: TokenType;      value: any;      firstCode?: CodeCharWrapper;    // 向下对应代码  } |

Token在词法单元的基本定义中拓展出了“firstCode”属性，用于存放该词法单元对应的第一个字符。这样，词法单元与代码间就建立了双向的引用，即既可通过代码寻找到对应的词法单元，也可通过词法单元找到对应的代码及其位置。这对于代码分析过程中的错误提示是十分有用的信息。

Token中的type是一个数字枚举或枚举属性，列举了所有合法的词法单元类型（亦称终结符）。终结符使用数字进行区分，其目的是为了与词法分析自动机的状态一一对应，因此该枚举内容较多，此处不宜置入，本文将在附录中置入TokenType枚举。

在前文中，我们将词法单元类型称为“终结符”，这跟文法中“终结符”的概念是基本一致的。对于语言来说，每种编程语言都有其对应的文法（用于识别同一语言的文法不唯一），文法由多条产生式组成。本工程中，文法产生式Production的数据结构如下所示：

|  |
| --- |
| export interface Production {      NT: string;         // 非终结符      T: Array<string>;   // 终结符  } |

在语法分析器中，给定C语言的简化版文法及其对应的LR表等信息后，系统会对词法单元序列进行语法分析，并产生语法树。语法树节点SyntaxNode的数据类型如下所示：

|  |
| --- |
| export interface SyntaxNode {      symbol: string;                 // 终结/非终结符      value?: any;                    // 终结符在语法分析阶段获得值，非终结符在执行阶段获得值      executeIndex?: number;          // 解释执行的过程中节点会被反复访问，每次访问的操作由访问次数决定      parent?: SyntaxNode;      children?: Array<SyntaxNode>;   // 非终结符的子树      token?: Token;                  // 向下对应词法单元  } |

其中，symbol是语法树节点的基本属性，value用于记录节点的值。作为一棵树，它可具有children。由于本系统的代码执行器是在语法树层面进行执行操作的，因此需要更多属性（executeIndex、parent）来辅助语法树遍历的操作。而token的作用则是向下对应词法单元，以便在语法分析和解释执行的过程中获取对应代码的位置。

使用文法生成语法树来描述语言是具有一定缺陷的，即同一层级的多个节点往往需要被描述成父子关系，这跟“将树或森林转换为二叉树”十分类似。从感官上来说，对于同级的内容，使用数组来表示会方便得多，也十分符合常识。因此在本工程中，使用了一种数据类型BaseNode去表示这种关系。BaseNode的主要作用是存储变量表，即对于每个程序块来说，都维护一份线性的变量表。除此以外，它还保存了对语法树节点SyntaxTree的引用。可以认为，语法树节点SyntaxTree是解释执行中的“站点”，而BaseNode则是一个随身携带的“记事本”与“传送门”。BaseNode数据结构如下所示：

|  |
| --- |
| export interface BaseNode {      variableList: Array<Variable>,      syntaxNode: SyntaxNode,      parentNode?: BaseNode,      subNode?: BaseNode,     // 可以是任何东西，包括 FunctionNode  } |

其余非核心部分的数据类型不在此做阐述，可直接通过代码中的注释了解他们的功能。

## 二、代码服务

代码服务（类名CodeService）负责为代码编辑器和编译器提供中间支持，是这两者之间承上启下且不可或缺的角色（在本模块中，“上”被定义为代码编辑器，“下”被定义为编译器）。

代码服务的主要功能是代码托管、代码定位、代码执行。下面将通过公共方法的方式说明代码服务的作用：

|  |
| --- |
| export interface CodeService {      getCodeLines(): Array<CodeLine>      getCodeLine(ln: number): CodeLine      getAllCode(): string      resetCode(content: string): void      insertCode(content: string, ln: number, col: number): CodePosition      deletePrevChar(ln: number, col: number): CodePosition      deleteNextChar(ln: number, col: number): CodePosition      deleteCodeLine(ln: number, col: number): CodeCharWrapper      readCharAt\_s(ln: number, col: number): CodeCharWrapper      readCharAt(ln: number, col: number): CodeCharWrapper      readNextChar(ln: number, col: number): CodeCharWrapper      readPrevChar(ln: number, col: number): CodeCharWrapper      getCodeToken(ln: number, col: number): Token      compile(): void      step(): void  } |

其中：

getCodeLines用于获取CodeService存储的所有代码行CodeLine。

getCodeLine用于获取单个指定行数的代码行。

getAllCode用于获取全部代码，即把所有内容都拼接为字符串输出。

insertCode用于在某个指定的位置插入代码。

deletePrevChar用于向前删除一位代码（退格键的一般功能）。

deleteNextChar用于向后删除一位代码（删除键的一般功能）。

deleteCodeLine用于删除一整行代码。

readCharAt用于读取指定位置的代码（用于处理用户光标操作，如果超出位置则自动取最接近的位置）。

readCharAt\_s用于读取指定位置的代码（用于代码扫描，如果超出位置则取出bof或eof）。

readNextChar用于读取指定位置的下一个代码。

readPrevChar用于读取指定位置的上一个代码。

compile用于对代码进行编译。用户在输入代码后，经过一定的延时，此函数将自动执行，用于为用户提供词法高亮与语法错误提示。

step用于以单步的方式运行程序。该函数将自动进行语法树的深度遍历，并在适当的时候停下来，以达到步进的功能。

## 三、页面视图

页面视图负责渲染控制程序的主体功能模块，包括菜单、图形演示区、代码编辑器三个部分。可以认为程序的中心是页面视图主控，而其它模块则为外挂组件。

程序启动后，首先会从index.tsx启动。该tsx引入了./containers/App.tsx，并以App作为ReactDOM的渲染入口，并将其挂载到HTML root元素上。

App组件初始化完成后，就会由React框架调用render函数，并在render函数中初始化菜单。图形演示区和代码编辑器则在选中教案之后才会开始渲染，并且由于选中教案后只能随时更换教案，而不能取消选择，因此无需考虑组件卸载。

### (一)菜单

菜单是程序初始化完成后首先进入的页面。该模块会引入教案加载器，获取教案目录，并将其显示在页面上。界面如图4.1所示。

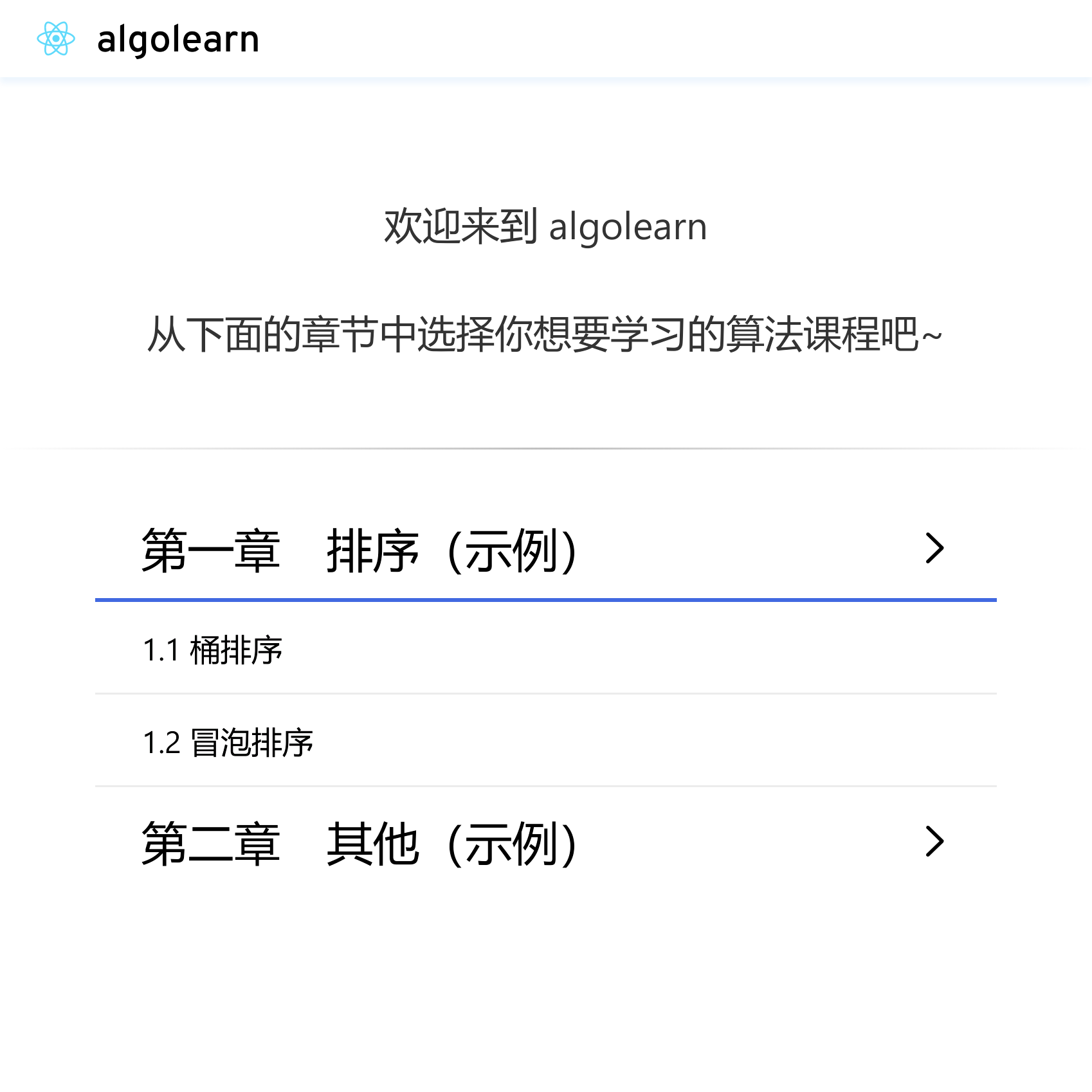


图4.1 菜单界面示例

选中任意一个教案后，菜单将会缩到页面左侧，露出右边的代码编辑器和中间的图形演示区。

### (二)图形演示区

图形演示区实际上自身并无可展示组件，这块区域的内容由教案加载器获得，并由主控进行加载和数据绑定。可以认为，图形演示区是一个单纯的“微型计算机”，教案加载器相当于插槽，需要往插槽中插入模块，才能在图形演示区显示出所需的内容。

### (三)代码编辑器

代码编辑器负责了代码输入、代码显示与词法高亮、语法错误指示、代码运行功能，是系统中的重要模块。它主要与代码服务保持通讯，将用户操作反馈到代码服务中去，并使用从代码服务取出的数据进行代码的渲染。

代码编辑器可分为以下组件：执行控制器、编辑器、边栏大小调节器。其中编辑器包含了代码显示、输入接收蒙层、光标指示器、语法错误指示器。用户在代码编辑器中的操作，实际上会被一个透明的蒙层截获，然后经过一系列的处理，将渲染结果显示在蒙层下方。而光标与语法错误指示器则悬浮在代码编辑器上方，用于指出对应的内容。代码编辑器界面如图4.2所示。



图4.2 代码编辑器界面示例

获取语法高亮颜色的相关代码也在代码编辑器CodeEditor中定义。如需要修改词法单元对应的颜色，可在CodeEditor中修改相关内容。

## 四、教案

教案，本系统中亦称呼“课程”，被设计成了独立的React模块，可根据不同的教学需求制作不同的教案；每个教案模块向外提供初始代码、预期变量表，向内要求提供程序的实际变量表，也可以依据实际需求添加不同的需求信息，如代码正在运行的行数等，然后通过这些信息渲染图形界面。

所有教案均继承自BaseCourse抽象对象。该抽象对象自身继承自React.Component，并在此基础上实现了非抽象方法componentDidMount，要求课程对象实现getBaseCode、getListeningVariable、onVariableChanged、render四个方法，相当于一个增加了功能和限制的React组件。只要使用React框架的编程方法即可轻松地为本系统添加课程。

BaseCourse抽象类定义如下所示：

|  |
| --- |
| export abstract class BaseCourse<Props = {}, State = {}> extends React.Component<Props, State> {      // 父组件需要在 props 里传入 onRef，这里组件挂载后自动调用，用于给父组件获取对该组件的 ref 引用      componentDidMount(): void {          (this.props as any).onRef(this);      }      abstract getBaseCode(): string;  // 示例代码      abstract render(): JSX.Element;  // render 函数  } |

## 五、教案加载器

教案加载器（淡橙色块）负责提供教案菜单，以及菜单项中对应的教案标题和教案模块。

每个“教案”都是一个CourseInfo对象。该对象的定义如下，其中title表示教案（节）标题，course表示BaseCourse教案模块。

|  |
| --- |
| export interface CourseInfo {      title: String;      course: React.ComponentClass<any, any>; // | typeof BaseCourse  } |

教案加载器自身硬编码了一个CourseMenu对象。该对象是一个ChapterInfo列表，存储了教案菜单以及CoueseInfo引用。ChapterInfo定义如下，其中title表示章标题，sections表示该章包含的节（教案）。

|  |
| --- |
| export interface ChapterInfo {      title: String;      sections: Array<CourseInfo>;  }; |

外界通过传入课程序号的方式获取相应的教案。教案不需要一个id进行区分，仅需要序号即可，因为主控在初始化程序时就会要求教案加载器返回课程列表。

课程序号CourseIndex被定义如下，其中chapter为章序号，section为节序号。

|  |
| --- |
| export interface CourseIndex {      chapter: number;      section: number;  } |

# 第五章 编译器模块详细设计

编译器用于对C语言程序进行编译操作，以为代码编辑器提供语法高亮、变量表、指令队列等功能。它是本系统的低层核心功能，包含了词法分析器和语法分析器。通过代码服务提供的功能扫描用户代码，在词法分析器中生成词法单元序列，词法单元序列通过语法分析器生成语法树，并在此过程中进行语法制导翻译和中间代码生成，产出指令队列。本章将介绍编译器的相关功能。

## 一、编译原理概述

程序设计语言是向人以及计算机描述计算过程的记号。在一个程序可以运行之前，它首先需要被翻译成一种可以被计算机执行的形式。完成这项翻译工作的软件系统称为编译器（complier）[13]。

简单地说，一个编译器就是一个程序，他可以阅读以某一种语言（源语言）编写的程序，并把该程序翻译成一个等价的、用另一种语言（目标语言）编写的程序[13]。参见图5.1。

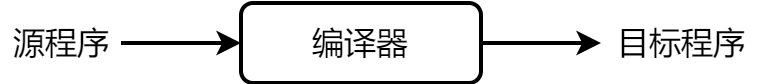


图5.1 一个编译器

解释器（interpreter）则是另一种常见的语言处理器。它并不生成目标程序，而是利用用户输入直接执行源程序[13]。参见图5.2。

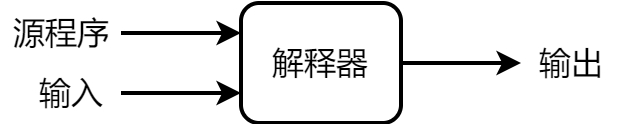


图5.2 一个解释器

## 二、编译器的结构

通过前面节的介绍，我们可以把编译器认为是一个黑盒子，它能够把源程序转换为目标程序。如果我们稍微把这个黑盒子打开一点，我们就可以看到这个过程实际是由两个部分组成：分析部分和综合部分。

分析（analysis）部分亦称之为编译器的“前端”（front end）。这个步骤将源程序分解成多种组成要素，组合成一种语法结构，然后使用这个语法结构来创建源程序的一种中间表示方法。这个步骤包含了错误检查过程，如果编译器发现源程序没有遵循正确的语法，或者语义上产生错误，它应能提供相应的信息以供用户进行改正。这个步骤往往会收集一些相关的信息，如表量表，或符号表，并将其与中间表示形式一并传送给综合步骤。

综合（synthesis）部分亦称之为编译器的“后端”（back end）。这个步骤根据分析步骤产生的中间表示形式和符号表中的信息构造相应的目标程序。

一些编译器在分析和综合两个步骤之间，还有一个与机器无关的代码优化步骤。它的目的是将中间表示形式上的程序进行一些转换，以达到更优的代码质量，如更小的内存需求、更少的代码数量、更快的执行速度等。

如果我们把一个完整的编译过程的步骤罗列出来，那么编译过程可分为词法分析、语法分析、语义分析、中间代码生成、机器无关代码优化、目标代码生成、机器相关代码优化。如图5.3所示。其中词法分析、语法分析、语义分析、中间代码生成为前端部分，目标代码生成和机器相关代码优化器为后端部分。

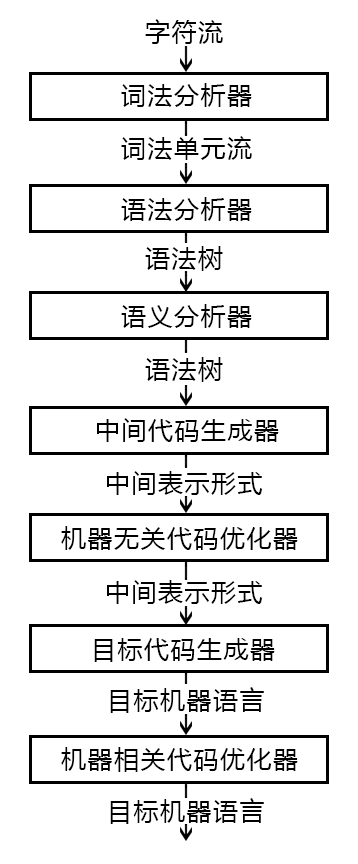


图5.3 完整编译过程的各个步骤

在本系统中，我们使用编译器的主要的目的并不是为了生成出一个可执行程序，而是为了获取符号表和中间代码（在本系统中称为“变量表”和“指令队列”），并对其进行解释执行。再者，该系统的目的是进行编程教学，因此我们可以忽略编译过程的很多步骤，仅需实现编译器的前端，而不需要通过编译器后端产出可执行程序。

我们可以用Java语言来类比本系统的编译操作。Java语言处理器结合了编译和解释过程，如图5.4所示[13]。Java源程序首先被编译成称为“字节码”（bytecode）的中间表示形式，然后由一个虚拟机对得到的字节码进行解释执行。它明确地拆出了编译过程的两大步骤，将机器无关的步骤前置进行，机器相关的步骤后置进行，使Java程序拥有了良好的可移植性。本系统包含了编译器前端和解释执行器两部分，由两个不同的模块负责，均在同一工程中进行。



图5.4 一个混合编译器

## 三、词法分析

编译的第一个步骤称为词法分析（lexical analysis）。词法分析器的主要任务是读入源程序的字符流，并将其组织成词法单元序列作为输出。词法分析器通常还要和符号表进行交互，当它发现了一个标识符的词素时，需要将这个词素添加到符号表中。

词法单元可以使用如下写法来表示：

<type, value>

比如我们要处理这样一个语句：

position = position + increment \* 16

经过词法分析器后，将会输出如表5-1的词法单元序列：

表5-1 position = position + increment \* 16 对应的词法单元序列

|  |  |
| --- | --- |
| **Type** | **value** |
| Identifier | “position” |
| Assign |  |
| Identifier | “position” |
| add |  |
| Identifier | “increment” |
| Multiply |  |
| number\_int | 60 |

为了将程序语句转换成词法单元序列，我们需要使用有穷自动机。

有穷自动机（Finite Automata, FA）是由两位神经物理学家MeCuloch和Pitts于1948年提出的，对一类处理系统建立的数学模型。这类系统具有一系列离散的输入输出信息和有穷数目的内部状态。系统只需要根据当前所处的状态和当前面临的输入信息就可以决定系统的后继行为。每当系统处理了当前的输入后，系统的内部状态也将发生改变。

例如，一个只接收“a”和“b”两种字符，并且以“abb”结尾的语言，其对应的一种自动机可使用如图5.5的转换图表示。

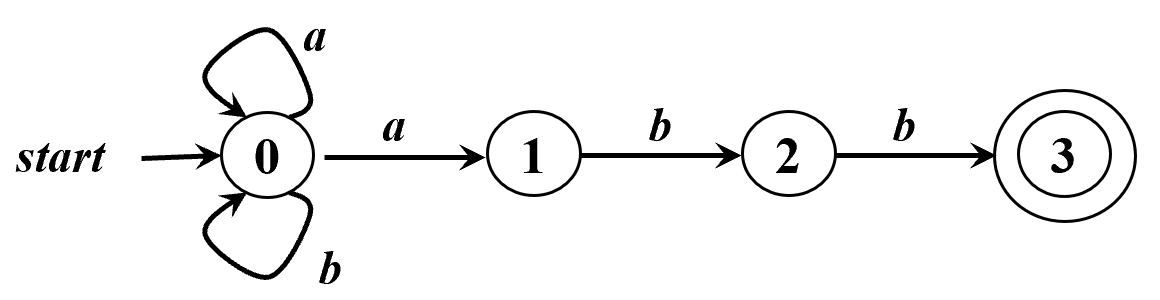


图5.5 一个状态转换图

本系统在进行词法分析时，使用了DFA。所使用的DFA对应的状态转换图如图5.6所示。

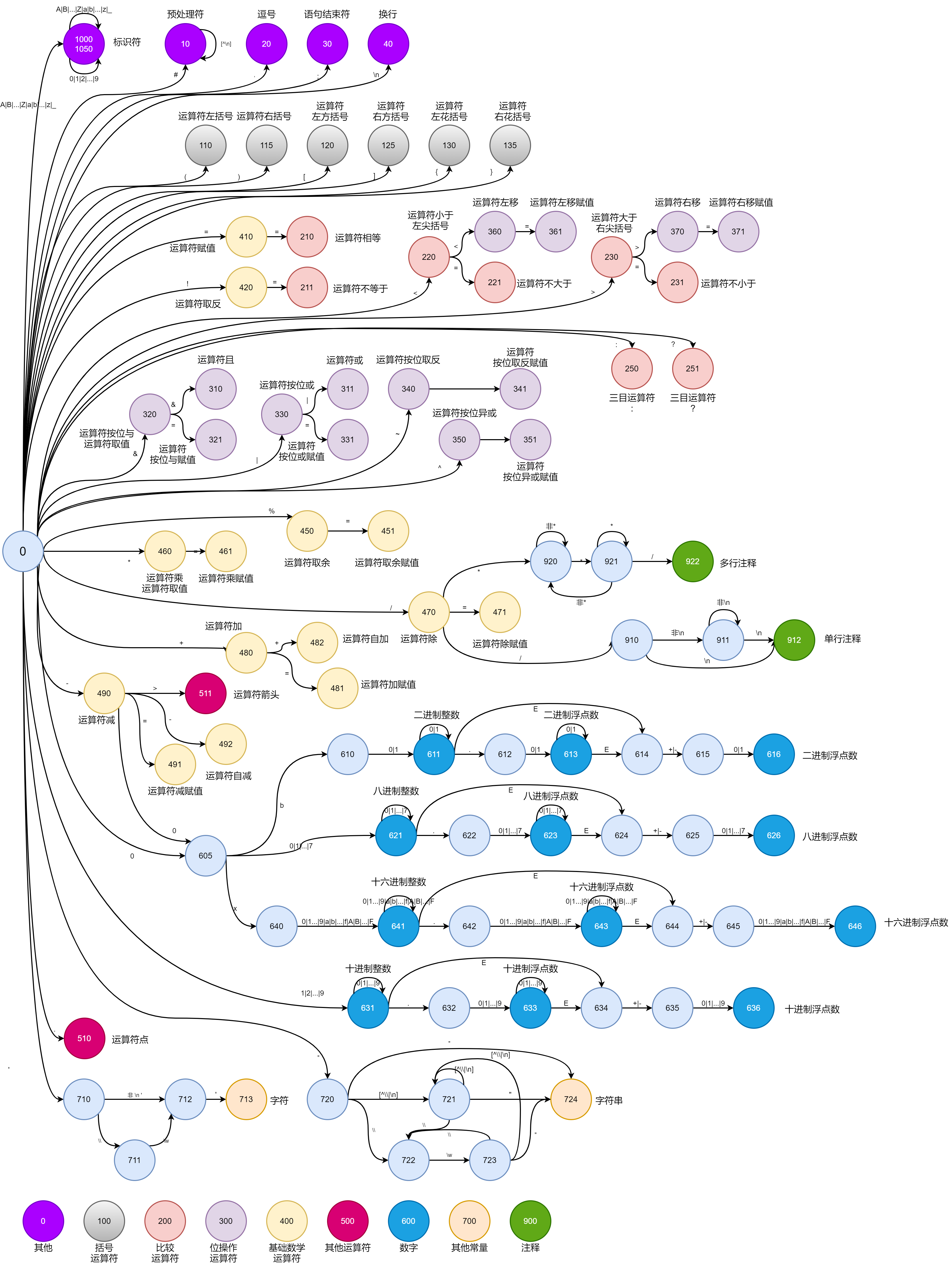


图5.6 本工程使用的C语言DFA

为了方便绘图，该状态转换图对标准定义的绘图方法进行了一定程度上的修改，见下述说明。

图中省略了“start”标识，规定0为DFA的开始状态。

所有非终结符均用淡天蓝色圆来表示。

所有终结符均用非淡天蓝色圆来表示，并且按照一定的分类模式对其进行了颜色标记，详见状态转换图底部的图例说明。

转移边的转移规则大体上使用正则表达式表示（正则表达式是一种使用正则文法的语言匹配方法，受篇幅限制，此处不进行其他描述，读者可自行查阅相关资料）。为了表述方便，进行了一些变形，如使用“非\n”表示可接收除了换行符以外的所有字符。转义字符使用JavaScript的转义符规则表示。

圆上的数字除了表示状态值以外，在终结符上还同时表示词法单元“type”枚举上的一个值。

当DFA进入“1000”状态时，表示接收到的单词可能是标识符，也可能是关键字。此时会进行一些额外分析，查找关键字是否包含该接收到的单词。如果有，则将其更改为对应的“type”枚举值。

DFA使用一个循环来读入程序中的每个字符。在一轮识别开始前，程序会先记录开始位置，然后开始走自动机。当DFA在某个状态接收到字符，但并没有与该字符在该状态下对应的转移边，则表示遇到了异常。此时DFA会检查遇到异常前的状态，如果是终结符，那么表示识别成功，并将前面识别的内容（从刚才标记的开始位置）全部标记为该终结符对应的词法单元类型枚举值，然后开始新一轮识别。如果不是终结符，则表示识别失败，出现了语法错误，将其类型标记为“错误”。

值得注意的是，本系统并没有完全实现C语言的所有语法特性。事实上，考虑到初学者往往不会使用太多复杂的语法，此DFA进行了较多的省略。同时，该词法分析器在获取词法单元值的时候，并不会按C语言的规则记录该变量的细节信息（如内存宽度等），而是按照JavaScript的类型系统进行记录。

词法分析服务由代码服务启动，并从代码服务中取得向后读取、向前读取等相关功能。一般情况下，在代码发生变化之后，代码服务会将此操作经过节流处理后，交由词法分析服务进行分析[14]。

## 四、语法分析

编译的第二个步骤称为语法分析（grammar analysis）。语法分析器的主要任务是从词法分析器获取一个由词法单元组成的串（词法单元序列），并验证这个串可以由源语言的文法生成，并同时构造出一颗语法分析树，并把它传递到编译器的其余部分进行进一步处理。事实上，语法分析器可能并不需要显式地构造出语法树，因为语法制导翻译和中间代码生成步骤可以在语法分析的过程中交错完成，因此，语法分析器和编译器前端的其他部分可以使用一个模块来实现。

编译器中常用的语法分析器分析方法可以分为自底向上和自顶向下的。顾名思义，自底向上的语法分析方法从分析树的底部（叶节点）向顶部（根节点）方向构造分析树，可以看成是将输入串w归约为文法开始符号S的过程；而自顶向下的语法分析方法从分析树的顶部（根节点）向底部（叶节点）方向构造分析树，可以看成是从文法开始符号S推导出词串w的过程。而这两种分析方法，语法分析器总是按从左到右的顺序扫描输入程序，每次扫描一个符号（词法单元）。

一般认为，自底向上的方法往往用于从句子识别语言，而自顶向下的方法往往用于从语言生成句子，因此本系统使用自底向上方法进行编程语言的识别，自顶向下方法此处不做讨论。

目前常用的自底向上语法分析器都基于LR(k)语法分析法的概念。其中“L”表示对输入进行从左到右的扫描，“R”表示反向构造出一个最右推导序列，“k”表示需要向前查看k个符号输入，一般情况下k≤1。本系统采用LR(1)分析技术对程序进行语法分析。

LR分析器实质上也是一种自动机。它具有两个栈：状态栈和符号栈；具有三种状态：移进、待约、归约；并具有动作表（ACTION）和转移表（GOTO），用于提供自动机的状态转移函数。

LR分析器在从左到右分析的过程中，每次过程都从剩余输入中读取一个符号，并且在ACTION表中寻找当前状态与当前输入匹配的动作。如“s1”表示使自动机跳转到状态4。然后，检查当前是否可以归约，即在ACTION表或GOTO表中将当前状态与当前输入匹配的操作，如“r1”表示使用文法规则的第1条进行归约操作。当最终操作为“acc”时，表示归约操作完成，如果此时剩余输入为空，那么分析成功。

比如，有这样一个文法：

|  |
| --- |
| **① *S*→*BB***  **② *B*→*aB***  **③ *B*→*b*** |

它的一个LR分析表如下表所示：

表5-2 一个LR分析表

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **状态** | **ACTION** | | | **GOTO** | |
| **a** | **b** | **$** | **S** | **B** |
| **0** | S3 | S4 |  | 1 | 2 |
| **1** |  |  | acc |  |  |
| **2** | S3 | S4 |  |  | 5 |
| **3** | S3 | S4 |  |  | 6 |
| **4** | R3 | R3 | R3 |  |  |
| **5** | R1 | R1 | R1 |  |  |
| **6** | R2 | R2 | R2 |  |  |

本系统在根据 C 语言的部分语法规则构造了一个适用于LR(1)语法分析器的简化版的 C 语言文法。该文法将在附录中给出。考虑到手工生成 LR 自动机和分析表工程量巨大，因此使用了自动化的 LR 分析器生成器。通过由博客园用户“Flowersea”提供的编译器前端生成代码[15]，给出本系统定义的简化版 C 语言文法，程序自动生成对应的LR表。将输出结果导入到程序中，即可实现 LR(1)语法分析器。

需要注意的是，为了方便，系统中的文法均使用小写字母加下划线的方式表示非终结符，使用大写字母及其它符号表示终结符。

## 五、变量表分析

编译程序的第三个步骤一般是语义分析。语义分析的功能是分析出语法树中每个节点的综合/继承属性值，判断代码是否符合语义规则，如某个变量只能拥有唯一的类型，如果在程序过程中给它进行了类型不匹配的赋值，则检出错误。该步骤对于编译执行的程序来说，往往是必要的。但对于解释执行的程序，该步骤有利于在编译过程即发现问题，但并非必要的。因此，本系统将“语义分析”简化为了“变量表分析”。即根据语法树生成出相应的变量表节点，以供解释执行的过程中使用。

存储变量表的数据结构为BaseNode，该数据类型的定义已在第四章中提及。此处不重复阐述。

事实上，根据系统需求，图形演示区往往只需要取函数内第一层的变量。因此，变量表分析器并不会把整个程序都分析一遍，而是仅分析函数第一层的变量表。在本系统中，为了存储函数特定的变量表结构，在BaseNode的基础上拓展出了ProgramNode和FunctionNode。其定义如下所示：

|  |
| --- |
| export interface ProgramNode extends BaseNode {      functionList: Array<FunctionNode>;  }  export interface FunctionNode extends BaseNode {      returnType: VariableType,      name: string,      parameterList: Array<Variable>,  } |

## 六、解释执行器

编译过程在变量表分析执行完成之后，基本工作就结束了。下一步操作便是根据用户需求执行程序代码。这一步操作将禁止用户再继续修改代码，同时应用到语法分析步骤产出的语法树和变量表分析步骤产出的变量表，对程序进行解释执行。

对程序进行解释执行的函数是代码服务中的step函数。该函数以语法树节点为基础，根据语法树的遍历历史决定执行什么操作，进入哪个节点。这个过程将是自动、连续的，循环过程中函数会判断非终结符的符号，如果是expression，那么就认为这是一个语句，在此跳出，等待用户操作，以达到单步执行的效果。如果遇到错误或者用户点击“重启”，则调用代码服务的reset函数，清除变量表，允许用户继续修改程序，并自动重新进行编译操作。

# 第六章 课程设计

教案是为用户提供教学的最终模块。在其他模块的支持下，教案模块可以向代码编辑器提供原始代码，监听程序状态，并以此来动态更新画面。

为方便用户理解，每个教案都被设计成文章的模式，动态部分使用虚线框表示，使用户知道此部分可以动态更新。事实上，由于每个教案的格式都是tsx，依靠强大的HTML5能力，动态内容可以实现远不止如此的效果。如果有特殊需求，动态内容甚至可以包括3D游戏。受限于开发时间和篇幅，此处只展示排序算法的部分相关内容。

## 桶排序

桶排序从“放置桌球”出发，以一个固定好尺寸的空间向用户展示如何使用桶排序对桌球进行排序。如图6.1所示。



图6.1 桶排序

## 二、插入排序

插入排序以上一节桶排序的结尾为基础，拓展出新的排序算法——在一个所需空间未知但不受限的地方内，如何在无需知道数字大小和数字数量的基础上，实现一个更符合人类直觉的排序算法——插入排序。如图6.2所示，右侧代码区域的变量与中间演示区相对应。如果缺失所需的变量或变量类型错误，则会给出相应的提示，如图6.3所示。



图6.2 插入排序  
  


图6.3 插入排序——错误显示

## 三、冒泡排序

冒泡排序基于人们生活中经常接触到的游戏——消消乐为作引入，介绍了冒泡的机制，以此来引出冒泡排序的算法。如图6.4所示。在课程的末尾，还举了按身高排队的例子来拓宽此种排序算法的实用性。



图6.4 冒泡排序

受限于开发时间和篇幅，其余算法的教案在此不做展示。

# 第七章 总结和展望

经过了几个月的努力，毕业设计终于顺利完成，也实现了一位合格的计算机专业本科生应有的实力水平的应用与展示。作为毕业前的最后一个大作品，它实现了我在大学期间，在学科部分的人生意义。

事实上，当初选中了这个毕业设计题目的时候，我并不是很乐意，因为任务书中的内容都与我所擅长的方向相去甚远——一方面，任务书中推荐使用python进行本系统的功能实现，但使用python实现算法类的可视化并没有现成的库，开发起来甚至需要涉及计算机图形学，深度颇大；而且我的常用语言并不包括python，对其相当陌生。另一方面，我主要擅长前端开发，这方面的开发并不需要应用到多复杂的算法，因此对于怎样去设计一个可适用于算法教学的案例库，我存在相当一部分的迷茫。

但幸好，我相信，尽管实现起来存在较大困难，但我心中对这套系统的实现方式已有了基本雏形，只要勇于挑战，还是能从中学到东西的。后来，经过会议讨论，我终于可以使用我所擅长的技术栈去完成这个系统了，于是我便选用了TypeScript，并以此来锻炼自己的系统设计与开发能力，为未来打下更扎实的基础。

一直以来，我对“计算机科学与技术”这个专业的实用性是存疑的，因为无论是根据我的日常开发经验，或者是实习经验，这个专业所学习的内容均与其无关，真正所需要使用到的知识，基本上全部来自于自主学习。我仅仅知道，这个专业带领我们从计算机底层的半导体器件开始，一直到高层的应用软件，一路探索了其实现方式，让我们领略了计算机学科的奥妙。但我们所学得的知识往往都已在现实生活中应用了超过十年，在这个技术快速发展的时代，这些过去的知识能在什么时候派上用场呢？

事实便是，它往往会在你意想不到的地方派上用场。一直以来，我对IDE（集成开发环境）所实现的自动化颇为惊叹。当我接触到实际的工作，把常用语言从JavaScript更换至TypeScript后，我便爱上了这款语言，因为它相对于语法松散的JavaScript来说，具有了更高的约束性，能搭配IDE实现更多令人惊叹的代码提示。它能自动地识别你程序代码中的所有元素，并在恰当的地方为你带来提示和自动补全，同时能验证程序是否存在语法性错误。这种高度自动化的工具，实际上便是电脑辅助人类生成代码的实际体现。而这一切的实现，底层实现便是编译原理。

我深知，所为一个科班生，我在这方面具有比其他培训班出来的程序员拥有更好的底子。而我曾经在知乎上阅览相关的话题时，亦看到了清华大学“实现了在自己写的CPU上运行自己写的操作系统，以及自己实现的编译器生成的程序”，甚至“三周造台计算机”的课程设计[16]。能将计算机原理体会得如此透彻并复现，正是这些学霸异于常人能力的体现，也是对计算机学科最好的诠释。我自然是远远不能达到这样的能力，我认为，只要能将学到的知识真正地应用起来，便是学习成果的体现。

事实上，无论是实现一台计算机、实现一个操作系统，还是实现一个编译器，都是相当不容易的事情。在实际开发的过程中，我便感受到了这个学科的困难之处。词法分析器实现起来较为简单，也符合直觉认知。但从语法分析开始，涉及到很多的概念，生成规则也较为复杂，生成规模更是随着设计规模的增加而指数上升，因此在设计这一块时，耗费了相当多的时间。万幸，我在网上找到了一个成功实现编译器前端的案例，并使用它的程序加上我的文法定义来生成语法分析器，为我节省了非常多的时间。不得不感叹——人类造工具的目的，就是为了用工具制造更好的工具，这也正是语言和编译器一路发展的历程。

在拿到毕业设计题目时，我已经踏进了互联网大厂实习的大门，因此可用于公司以外的项目开发的时间是少之又少。即使是我更喜欢的另一个自主作品——FFBox，在数个月期间也只是贡献了寥寥几百行代码，更别谈利用闲暇时间做毕业设计。因此我一早就向组长汇报了长期请假的计划，并在四月中旬开始请假。尽管如此，我依然没有腾出足够的时间进行毕业设计的制作，因此很多功能都十分简陋，教案的数量也相当少。但我相信，我的这份毕业设计是一个有意义的作品。如果能在后期加以完善和改造，在投入实际使用之后，必将为计算机算法的初级学习者带来便利，为国家培养更多的计算机人才。

# 参考文献

[1]　　TIOBE Index [OL]．https://www.tiobe.com/tiobe-index/

[2]　　TypeScript中文网 [OL]．https://www.tslang.cn/index.html

[3]　　网景 - 维基百科 [OL]．https://zh.wikipedia.org/wiki/%E7%B6%B2%E6%99%AF

[4]　　TC39 – Ecma International [OL]．https://www.ecma-international.org/technical-committees/tc39/

[5]　　JavaScript - 维基百科 [OL]．https://zh.wikipedia.org/wiki/JavaScript

[6]　　React官方中文文档 [OL]．https://zh-hans.reactjs.org/

[7]　　Babel中文网 [OL]．https://www.babeljs.cn/

[8]　　webpack [OL]．https://webpack.docschina.org/

[9]　　Vue CLI [OL]．https://cli.vuejs.org/zh/

[10]　　facebook/create-react-app - github [OL]．https://github.com/facebook/create-react-app

[11]　　Node.js中文网 [OL]．http://nodejs.cn/

[12]　　About npm | npm docs [OL]．https://docs.npmjs.com/about-npm

[13]　　Alfred V. Aho et al，赵建华、郑滔、戴新宇译．编译原理 第2版 [M] ．机械工业出版社，2009.1

[14]　　安歌．浅谈 JS 防抖和节流 [OL]．https://segmentfault.com/a/1190000018428170

[15]　　FlowerSea．LR(1)分析表-语法树-四元式[OL]．<https://www.cnblogs.com/baocong/p/9242321.html>

[16]　　思佳、苑洁、思彤，清华大学．三周造台计算机！不难？！ [OL]． https://mp.weixin.qq.com/s/RBLvl-ZpDQm\_RBG4DJgt\_g

# 附　　录

## 一、相关代码

### (一) TokenType枚举值（对应词法分析自动机状态值）

|  |
| --- |
| export enum TokenType {      error = -1,         // 状态机终态或非终态过后读取到的无法转移状态      unknown = 0,        // 未读取      preprocess = 10,    // 预处理（整句）      comma = 20,         // 逗号      semicon = 30,       // 分号      endline = 40,       // 换行      brakets\_round\_left = 110,   // (  左圆括号      brakets\_round\_right = 115,  // )  右圆括号      brakets\_square\_left = 120,  // [  左方括号      brakets\_square\_right = 125, // ]  右方括号      brakets\_curly\_left = 130,   // {  左花括号      brakets\_curly\_right = 135,  // }  右花括号      compare\_equal = 210,        // == 相等      compare\_unequal = 211,      // != 不相等      compare\_less = 220,         // <  小于      compare\_less\_equal = 221,   // <= 小于等于      compare\_great = 230,        // >  大于      compare\_great\_equal = 231,  // >= 大于      compare\_colon = 250,        // :  三目——冒号      compare\_question = 251,     // ?  三目——问号      bit\_logic\_and = 310,        // && 逻辑与      bit\_logic\_or = 311,         // || 逻辑或      bit\_and = 320,              // &  按位与      bit\_and\_assign = 321,       // &= 按位与赋值      bit\_or = 330,               // |  按位与      bit\_or\_assign = 331,        // |= 按位或赋值      bit\_negation = 340,         // |  按位取反      bit\_negation\_assign = 341,  // |= 按位取反赋值      bit\_xor = 350,              // |  按位异或      bit\_xor\_assign = 351,       // |= 按位异或赋值      bit\_move\_left = 360,        // << 位左移      bit\_move\_left\_assign = 361, // <<= 位左移赋值      bit\_move\_right = 370,       // >> 位右移      bit\_move\_right\_assign = 371,// >>= 位右移赋值      calc\_assign = 410,          // =  赋值      calc\_negation = 420,        // !  取反      calc\_mod = 450,             // %  取余      calc\_mod\_assign = 451,      // %= 取余赋值      calc\_multiply = 460,        // \*  乘      calc\_multiply\_assign = 461, // \*= 乘赋值      calc\_devide = 470,          // /  除以      calc\_devide\_assign = 471,   // /= 除以赋值      calc\_add = 480,             // +  加      calc\_add\_assign = 481,      // += 加赋值      calc\_add\_self = 482,        // ++ 自加      calc\_minus = 490,           // -  减      calc\_minus\_assign = 491,    // -= 减赋值      calc\_minus\_self = 492,      // -- 自减      struct\_point = 510,         // .  结构体点操作      struct\_arrow = 511,         // -> 结构体箭头操作      number\_bin\_int = 611,       // 二进制整数      number\_bin\_float = 613,     // 二进制浮点数      number\_bin\_float\_e = 616,   // 二进制浮点数带 E      number\_oct\_int = 621,       // 八进制整数      number\_oct\_float = 623,     // 八进制浮点数      number\_oct\_float\_e = 626,   // 八进制浮点数带 E      number\_dec\_int = 631,       // 十进制整数      number\_dec\_float = 633,     // 十进制浮点数      number\_dec\_float\_e = 636,   // 十进制浮点数带 E      number\_hex\_int = 641,       // 十六进制整数      number\_hex\_float = 643,     // 十六进制浮点数      number\_hex\_float\_e = 646,   // 十六进制浮点数带 E      char\_char = 713,            // '' 字符      char\_string = 724,          // "" 字符串      bool\_true = 730,            // 布尔真      bool\_false = 731,           // 布尔假      note\_singleline = 912,      // // 单行注释      note\_multiline = 922,       // /\* \*/ 多行注释      identifier = 1000,          // 标识符      keyword = 1100,             // 关键字      keyword\_void = 1101,      keyword\_short = 1102,      keyword\_int = 1103,      keyword\_long = 1104,      keyword\_float = 1105,      keyword\_double = 1106,      keyword\_char = 1107,      keyword\_for = 1108,      keyword\_while = 1109,      keyword\_do = 1110,      keyword\_if = 1111,      keyword\_else = 1112,      keyword\_switch = 1113,      keyword\_case = 1114,      keyword\_break = 1115,      keyword\_continue = 1116,      keyword\_return = 1117,      eof = 9999,                 // 文件结尾      bof = 9998,                 // 文件开头第 -1 个字符（无效位置）  } |

### (二) C语言简化版文法

|  |
| --- |
| program' -> program  program -> function\_definition\_list  program -> variable\_definition\_list function\_definition\_list  function\_definition\_list -> function\_definition  function\_definition\_list -> function\_definition function\_definition\_list  function\_definition -> variable\_type function\_declaration compound\_statement  variable\_type -> VOID  variable\_type -> INT  variable\_type -> FLOAT  array\_closure -> [ NUM ] array\_closure  array\_closure -> [ NUM ]  array\_expression\_closure -> [ equality\_expression ] array\_expression\_closure  array\_expression\_closure -> [ equality\_expression ]  function\_declaration -> ID ( )  function\_declaration -> ID ( parameter\_list )  parameter\_list -> parameter\_declaration  parameter\_list -> parameter\_declaration , parameter\_list  parameter\_declaration -> variable\_type ID  parameter\_declaration -> variable\_type ID array\_closure  compound\_statement -> { }  compound\_statement -> { variable\_definition\_list }  compound\_statement -> { statement\_list }  compound\_statement -> { variable\_definition\_list statement\_list }  variable\_definition\_list -> variable\_definition  variable\_definition\_list -> variable\_definition variable\_definition\_list  variable\_definition -> variable\_type ID ;  variable\_definition -> variable\_type ID array\_closure ;  statement\_list -> statement  statement\_list -> statement statement\_list  statement -> compound\_statement  statement -> expression\_statement  statement -> selection\_statement  statement -> iteration\_statement  statement -> jump\_statement  expression\_statement -> ;  expression\_statement -> expression ;  selection\_statement -> IF ( expression ) statement  selection\_statement -> IF ( expression ) statement ELSE statement  iteration\_statement -> WHILE ( expression ) statement  jump\_statement -> RETURN ;  jump\_statement -> RETURN expression ;  expression -> assignment\_expression  expression -> assignment\_expression , expression  assignment\_expression -> addressing\_expression = assignment\_expression  assignment\_expression -> logical\_expression  assignment\_expression -> addressing\_expression  addressing\_expression -> ID  addressing\_expression -> ID array\_expression\_closure  logical\_expression -> equality\_expression  logical\_expression -> equality\_expression && logical\_expression  logical\_expression -> equality\_expression || logical\_expression  equality\_expression -> relational\_expression  equality\_expression -> relational\_expression == equality\_expression  equality\_expression -> relational\_expression != equality\_expression  relational\_expression -> additive\_expression  relational\_expression -> additive\_expression < relational\_expression  relational\_expression -> additive\_expression > relational\_expression  relational\_expression -> additive\_expression <= relational\_expression  relational\_expression -> additive\_expression >= relational\_expression  additive\_expression -> multiplicative\_expression  additive\_expression -> multiplicative\_expression + additive\_expression  additive\_expression -> multiplicative\_expression - additive\_expression  multiplicative\_expression -> unary\_expression  multiplicative\_expression -> unary\_expression \* multiplicative\_expression  multiplicative\_expression -> unary\_expression / multiplicative\_expression  unary\_expression -> primary\_expression  unary\_expression -> unary\_operator unary\_expression  primary\_expression -> addressing\_expression  primary\_expression -> NUM  primary\_expression -> function\_call  primary\_expression -> ( expression )  unary\_operator -> +  unary\_operator -> -  function\_call -> ID ( )  function\_call -> ID ( equality\_expression\_list )  equality\_expression\_list -> equality\_expression  equality\_expression\_list -> equality\_expression , equality\_expression\_list |

## 二、相关资料中英文对照

### (一) 英文原文

**THE INTRODUCTION TO TYPESCRIPT**

TypeScript is a programming language developed and maintained by Microsoft. It is a strict syntactical superset of JavaScript and adds optional static typing to the language. TypeScript is designed for the development of large applications and transcompiles to JavaScript. As TypeScript is a superset of JavaScript, existing JavaScript programs are also valid TypeScript programs.

TypeScript may be used to develop JavaScript applications for both client-side and server-side execution (as with Node.js or Deno). There are multiple options available for transcompilation. Either the default TypeScript Checker can be used, or the Babel compiler can be invoked to convert TypeScript to JavaScript.

TypeScript supports definition files that can contain type information of existing JavaScript libraries, much like C++ header files can describe the structure of existing object files. This enables other programs to use the values defined in the files as if they were statically typed TypeScript entities. There are third-party header files for popular libraries such as jQuery, MongoDB, and D3.js. TypeScript headers for the Node.js basic modules are also available, allowing development of Node.js programs within TypeScript.

The TypeScript compiler is itself written in TypeScript and compiled to JavaScript. It is licensed under the Apache License 2.0. TypeScript is included as a first-class programming language in Microsoft Visual Studio 2013 Update 2 and later, alongside C# and other Microsoft languages.An official extension allows Visual Studio 2012 to support TypeScript as well. Anders Hejlsberg, lead architect of C# and creator of Delphi and Turbo Pascal, has worked on the development of TypeScript.

**History**

TypeScript was first made public in October 2012 (at version 0.8), after two years of internal development at Microsoft. Soon after the announcement, Miguel de Icaza praised the language itself, but criticized the lack of mature IDE support apart from Microsoft Visual Studio, which was not available on Linux and OS X at that time. As of April 2021 there is support in other IDEs and text editors, including Emacs, Vim, Webstorm, Atom and Microsoft's own Visual Studio Code.

TypeScript 0.9, released in 2013, added support for generics. TypeScript 1.0 was released at Microsoft's Build developer conference in 2014. Visual Studio 2013 Update 2 provides built-in support for TypeScript.

In July 2014, the development team announced a new TypeScript compiler, claiming 5× performance gains. Simultaneously, the source code, which was initially hosted on CodePlex, was moved to GitHub.

On 22 September 2016, TypeScript 2.0 was released; it introduced several features, including the ability for programmers to optionally prevent variables from being assigned null values, sometimes referred to as the billion-dollar mistake.

TypeScript 3.0 was released on 30 July 2018, bringing many language additions like tuples in rest parameters and spread expressions, rest parameters with tuple types, generic rest parameters and so on.

TypeScript 4.0 was released on 20th, August 2020. While 4.0 did not introduce any breaking changes, it added language features such as Custom JSX Factories and Variadic Tuple Types.

**Design**

TypeScript originated from the shortcomings of JavaScript for the development of large-scale applications both at Microsoft and among their external customers. Challenges with dealing with complex JavaScript code led to demand for custom tooling to ease developing of components in the language.

TypeScript developers sought a solution that would not break compatibility with the standard and its cross-platform support. Knowing that the current ECMAScript standard proposal promised future support for class-based programming, TypeScript was based on that proposal. That led to a JavaScript compiler with a set of syntactical language extensions, a superset based on the proposal, that transforms the extensions into regular JavaScript. In this sense TypeScript was a preview of what to expect of ECMAScript 2015. A unique aspect not in the proposal, but added to TypeScript, is optional static typing that enables static language analysis, which facilitates tooling and IDE support.

### (一) 中文翻译

TypeScript是Microsoft开发和维护的一种编程语言。它是JavaScript的严格语法超集，并为该语言添加了可选的静态类型。TypeScript是为开发大型应用程序而设计的，并可以将其转换为JavaScript。由于TypeScript是JavaScript的超集，因此现有的JavaScript程序也是有效的TypeScript程序。

TypeScript可用于开发JavaScript应用程序，以用于客户端和服务器端执行（与Node.js或Deno一样）。有多个可用于反编译的选项。可以使用默认的TypeScript Checker，或可以调用Babel编译器将TypeScript转换为JavaScript。

TypeScript支持可以包含现有JavaScript库的类型信息的定义文件，就像C ++ 头文件可以描述现有目标文件的结构一样。这使其他程序可以使用文件中定义的值，就像它们是静态键入的TypeScript实体一样。有一些流行的库（例如jQuery，MongoDB和D3.js）的第三方头文件。还提供了Node.js基本模块的TypeScript标头，从而允许在TypeScript中开发Node.js程序。

TypeScript编译器本身是用TypeScript编写的，并已编译为JavaScript。它是根据Apache License 2.0许可的。TypeScript与C＃和其他Microsoft语言一起作为Microsoft Visual Studio 2013 Update 2和更高版本中的一流编程语言提供。官方扩展允许Visual Studio 2012也支持TypeScript。C＃的首席架构师，Delphi和Turbo Pascal的创建者Anders Hejlsberg致力于TypeScript的开发。

**历史**

在经过两年的内部开发之后，TypeScript于2012年10月首次公开（版本为0.8）。消息宣布后不久，Miguel de Icaza赞扬了该语言本身，但批评说除了Microsoft Visual Studio之外，缺乏成熟的IDE支持，该支持当时在Linux和OS X上不可用。截至2021年4月，其他IDE和文本编辑器均提供支持，包括Emacs，Vim，Webstorm，Atom 和Microsoft自己的Visual Studio Code。

2013年发布的TypeScript 0.9增加了对泛型的支持。TypeScript 1.0在2014年的Microsoft Build开发者大会上发布。 Visual Studio 2013 Update 2提供了对TypeScript的内置支持。

2014年7月，开发团队发布了新的TypeScript编译器，声称其性能提高了5倍。同时，最初托管在CodePlex上的源代码已移至GitHub。

2016年9月22日，发布了TypeScript 2.0；它引入了几个功能，包括使程序员有选择地防止为变量分配null值的功能，有时被称为十亿美元的错误。

TypeScript 3.0于2018年7月30日发布带来了许多语言新增功能，如rest参数和扩展表达式中的元组，具有tuple类型的rest参数，通用rest参数等。

TypeScript 4.0于2020年8月20日发布。4.0没有带来任何重大变化，但它增加了诸如自定义JSX工厂和可变元组类型的语言功能。

**设计**

TypeScript起源于JavaScript在Microsoft及其外部客户中开发大型应用程序的缺点。处理复杂的JavaScript代码带来的挑战导致需要定制工具来简化语言组件的开发。

TypeScript开发人员寻求一种不破坏与标准及其跨平台支持的兼容性的解决方案。知道当前的ECMAScript标准建议书有望为将来的基于类的编程提供支持，因此TypeScript就是基于该建议书的。这导致JavaScript编译器具有一组语法语言扩展，这是基于该提案的超集，可将扩展转换为常规JavaScript。从这个意义上讲，TypeScript是ECMAScript 2015预期的预览。提案中未包括但未添加到提案中的一个独特方面是可选的静态类型，该类型可启用静态语言分析，从而有助于工具和IDE支持。

# 致　　谢

论文写到致谢，一时间不知如何开口，遂至网上查阅广大学子肺腑之言。

可惜，他们或是普普通通流水式致谢，或是花式作文只是文科生的把戏，或是致敬、回忆、道别大学四年的青春。

可是为什么要那样隆重呢？是因为毕业过后就再也无法相见？是因为日后就再也不能回来此处？还是写这份毕业设计耗尽了一生所能，为一生之绝唱？

我躺上床，睡了一觉，我觉得我不是这样的人。

现代科技如此发达，何忧道别以后不能再相见？这可与这篇论文没有太大关系。

那又为什么需要在此处致谢四年的青春呢？我宁愿不希望青春如此就结束，而是能把这份精神延续下去。

我们都有光明的未来，我们未来可以写出比现在更优秀的代码、更优秀的作品。这不是我的绝唱，只是一个高点。这个高点证明了我终于从一个对计算机专业懵懂无知的少年，成长为了能利用它使自己发光发亮的少年。

因此，我首先感谢我自己，为了这份毕设设计所付出的数个月的努力。为此，我掌握了计算机系学生的专属课程，了解到了计算机的奥秘。虽然我并不能指出它在未来有什么特定的作用，但我相信，学到的就是自己的，科技总能为人带来力量。

同时，也要感谢学院里所有老师为我们带来的学习平台、资源和环境。在校期间，上过那么多老师的课程，很难说出是哪一位特定的老师成就了今天的我，但可以肯定的是，王老师作为这份毕业设计的导师，成就了这份作品。正因为有这样一份探索的心，提出了这样的设想，促使我将其实现。得益于老师在我进行制作的时候对系统设想的足够肯定，我对这份作品充满信心。

当然，除此以外，还有一位场外人员是值得感谢的。尽管素未谋面，甚至不曾说一句话，但他确实为我解铺平了毕设过程中的一座大山。那就是一位博客园用户，Flowersea。在遇到他的文章之前，我不知道为什么会有“为什么不建议手写LR(1)分析器”的提问，直到使用他的程序为我自动生成了语法分析器，我才终于理解，并深感庆幸。原来老师说的“实现一个C语言的LR(1)分析器，会产生数千个状态”，是真的。幸好有如此的自动化工具，实在是为我节省了大量时间。

或许还会有其他人，我心存感谢，但一时未能想起来，深感抱歉。但大家的付出，我看在眼里，放在心上，便不在此过多提及了。

感谢大家！