目　录

1 绪论 2

1.1 研究背景与意义 2

1.2 国内外研究现状 2

1.3 研究的主要内容 4

1.4 论文的组织结构 5

2 系统分析 5

2.1 需求分析 5

2.2 可行性分析 6

3 相关技术介绍 7

3.1 LayUI 7

3.2 Cprase 7

3.3 LR 分析法 7

3.4 Libevent 8

3.5 RapidJSON 9

3.6 分离式编译模式 9

4 系统概要设计 11

4.1 总体架构设计 11

4.2 功能模块设计 11

5 系统功能详细设计与实现 14

5.1 代码编译模块 14

5.1.1 流程设计 14

5.1.2 核心代码 18

5.1.3 实现效果 19

5.2 请求处理模块 23

5.2.1 流程设计 23

5.2.2 核心代码 26

5.2.3 实现效果 27

5.3 可视化解析模块 29
5.3.1 流程设计 29
5.3.2 核心代码 32
5.3.3 实现效果 33
5.4 动态渲染模块 34
5.4.1 流程设计 34
5.4.2 核心代码 35
5.4.3 实现效果 35
5.5 系统算法库及其动态扩展 40
5.5.1 系统算法库 40
5.5.2 算法库的动态扩展 43
6 总结与展望 46
6.1 研究总结 46
6.2 未来展望 47
参考文献 49

可视化算法交互式平台的设计与实现

摘要：由于算法本身十分抽象，学生学习算法以及老师讲述算法都往往感到吃力。目前普遍使用 PPT 来阐述算法，虽然有演示但只是执行程序展示结果，同时，由于 PPT 的制作水平有限，很多算法演示主要以展示知识为主，教师只能按照设定的演示顺序进行教学，因此，迫切地需要一个能将算法执行过程动态展示的平台。在深入研究了代码的编译过程之后，系统提出了一种先编译后可视化的解决方案，可视化算法交互式平台实现了在线编译代码的功能，再在此基础上动态渲染算法代码，而且具备一定的交互能力，以期用生动有趣的动态效果来阐述深奥难懂的算法原理。

关键词：算法与数据结构；在线编译；可视化

Design and Implementation of Interactive Platform for Visualization Algorithm

Abstract: Because the algorithm itself is very abstract, the student learning algorithm and the teacher's narrative algorithm are often difficult. Although there is a demonstration but only the execution program shows the results, and because the production level of the powerpoint is limited, many of the algorithm demonstrations are mainly based on the display knowledge, and the teacher can only teach in the order of the set, so it urgently needs a platform that can dynamically display the algorithm. After studying the compilation process of the code, this system presents a pre-compiled solution, and the visual algorithm interactive platform implements the function of online compiling code, and then dynamically rendering the algorithm code on this basis, and has the ability to interact, in order to illustrate the principle of abstruse algorithm with vivid and interesting dynamic effect.

Key words: Algorithm and data structure; Online compilation; Visualization

1 绪论

1.1 研究背景与意义

数据结构与算法是计算机专业及其相关专业的一门必修课，也是核心课程之一[1]，学好它也计算机专业后续的深入学习的基础[2]。但是，对于算法的学习，这本身是十分抽象而且难度颇高的，算法课程需要学生理解算法的运行机理，目前对于算法教学领域普遍采用 PPT 讲授的形式[3]，虽然有演示但只是执行程序展示结果，忽略了计算机系统内部算法数据的演示和算法运行的动态过程，学生无法和演示过程进行交互[4]，影响学生对于算法本质的理解，可以简单的演示算法的一些工作流程，但是仅仅是通过幻灯片和图片演示没有提供和学生交流互动的平台[5]，很难让学生参与其中实际操作，这大大降低了大家的学习兴趣，而且，对于算法的动态演示来说，由于制作多媒体课件水平有限，好多算法演示的多媒体课件主要以展示知识为主，教师只能按照设定的演示顺序进行教学，这也不利于算法的教学，学生看到多媒体动画根本没时间细细思索[6]，而动态可视化比如计算机软件、动画和交互式技术很少使用到，这一定程度上降低了多媒体教学的可视化效果，难以有效的展示数据结构和算法的抽象性和复杂性。

算法的可视化是将一个程序的数据、操作和语义提取出来并进行动态演示的过程，利用诸如文本、颜色、声音、图形、编码、动画和视频等多媒体工具集合来描述算法。设计算法可视化程序可以使算法的执行过程更直观，也使得算法的思想以一种更简单的动态效果呈现出来。算法可视化一般分为两类：（1）静态算法可视化，通过用一系列静止的图像来展示算法的执行过程；（2）动态算法可视化，即动画算法，这是一种类似于电影的连续地描述算法的执行过程[7]。算法可视化的最直接用处就是辅助教学，算法的教学难点在于它们的抽象性和动态性。在书本教材和课堂授课板书或投影胶片中接受图示可以在肯定程度上化抽象为直观，但很难呈现对象的瞬间动态特性和算法的作用过程。

综上所述，对于在教学中引入更加方便、易操作、扩展性好的算法可视化平台等研究都是有十分重大意义的。

1.2 国内外研究现状

可视化系统是计算机技术的应用领域之一，在可视化技术日渐成熟的背景下，专家们将该技术应用于开发数据结构与算法演示系统。目前，大多数算法演示系统的思想都源自 Brown 等提出的 BALSA 系统。该系统首次提出感兴趣事件的概念，用于标识算法状态改变的关键点，只有对这种关键点进行可视化义才更有意义。目前，领域内已涌现出大量的算法演示系统，如 POLKA、ALADDIN、GAIGS 等等。多媒体、可视化以及丰富的其他

表现方式已成为最近所关注的焦点。

国内的可视化程序的教学起步较晚，算法可视化程序也是较少的，大多数的算法可视化程序都正处于研究阶段。目前国内的算法可视化的教学研究基本上可以归纳为两种：第一种是动画演示，即采用播放视频或者幻灯片的形式来演示算法的工作过程；第二钟是可视化调试，即根据编写好的程序代码，采用调试程序来理解算法的思想[8]。起初在国内做的比较好的有应用到教学中案例，知名度比较高的就是在清华大学的严蔚敏教授编著的《数据结构》系列教材中，里面有着算法演示软件的配套可视化软件，该软件有 PASCAL 和 C 语言两种版本的解释，实现了数据结构书籍中的大部分典型算法的演示，数据结构算法本身是十分抽象的，但是可以通过动态图像的方式展示出来，在演示时，可以查看配套语言的中间变量的数据变化，随后虽然关于算法与数据结构的书籍逐渐增多，也出来了一些类似的可视化软件，但是仍然没有太大改善，绝大多数也停留于展示执行过程与结果的阶段[9]。

国外对于算法可视化的研究比较早，也有很多优秀的可视化工具。国外算法可视化系统大概有如下特点：他们让学生输入数据，或者提供一个预先准备好的数据集；他们能跟踪算法在数据集上的工作流程或动画的形式播放或分步演示。随着算法的运行，数据的视觉表示可以被修改和更新。一些系统还提供伪代码，让学生看每一行代码是如何作用于数据，并提供了机制系统的一些历史记录，学生可以复习以前的实施步骤。国外使用最广的开发算法可视化的方法是通过脚本语言模拟算法并且产生可视化效果，比如 VisuAlgo、Illustrated Algorithms、Applet Demonstration Site、Visualizing Algorithms、Sorting Algorithms 之类的可视化系统[10]。

从实现技术看，现有的数据结构和算法演示系统可以分为下面两类：（1）动态演示构件：此类演示构件是利用一些软件工具制作的，例如 Flash、PowerPoint、Authorware 等。利用它们制作的演示构件，画面华丽，而且制作方法简单。但是由于该方法依赖于软件工具，因此实现方式受到限制，难以恰如其分地演示数据结构和算法的动态过程，而且在交互性上具有较大缺陷。（2）可视化系统可视化系统对算法演示界面的布局较为统一，系统演示界面一般分为三个部分，分别是动画演示区（包括算法的图形变换和辅助演示标志）、代码演示区、数据提示区三个部分。每个部分既相互独立又相互联系，它的算法演示过程、代码显示过程和数据提示，使得学习者可以观察到算法的执行流程以及执行过程中数据的变化，有利于学习者对算法的理解。

可视化系统使用的确定程序中可视化的技术通常可以分为四种类型：（1）事件驱动型。

事件驱动方法是指在程序需要演示的程序点用感兴趣事件进行注释，然后根据这些感兴趣事件进行相应演示动画的编辑，它采用事件驱动来实现可视化的先驱系统有 BALSA，以及已经应用于许多算法动态演示系统的 Zeus。事件驱动的特点是需要重新编写算法源代码和编写动画的源代码，这增加了用户使用系统的难度。（2）状态驱动型。状态驱动方法是指在程序与可视化状态之间确定一个映射关系，这个映射关系通常在程序运行之前由可视化工具指定。状态驱动方法的特点是将编写动画的部分从源代码中分离，但是在使用这些系统的时候，用户还是需要重新编写程序源代码，从计算状态到图形对象的映射是相当复杂的，而且如果图形对象不适用，用户还需要重新设计一个新的图形对象。（3）自动动画型。自动动画方法是开发算法演示系统最简单的方法，它提高了程序开发人员的编程效率，而且这种方法调试起来十分方便。（4）可视程序设计型。可视程序设计方法在于通过可视符号表示程序命令和句子来使程序更容易确定，使用可视程序设计方法的优点是需要可视化对象的路径与属性的细节都由系统自动提供，开发人员只需要关心动画的高层次描述，也就是说，开发人员只需要关心什么对象需要可视化，而不用关心如何对这些对象可视化。

通过对上述几类演示系统的比较以及可视化技术的研究，可以发现，可视化系统具备更好的交互性，但是相对地对开发人员的要求也更高。虽然现今开发出来的数据结构与算法演示系统数量众多，质量也很高，但是都无法避免地在交互性方面具有一些瑕疵。

1.3 研究的主要内容

之前的许多可视化算法平台，大多只是完成了“可视化”的功能，仅仅展示了对应算法的执行过程与结果，甚至执行的代码都不可见，这样的平台对于算法的用户而言，只看到了“开始”与“结果”，未能参与真正的“学习算法的过程”，这样也就不完全满足辅助教学的需求。所以，可视化算法交互式平台没有简简单单停留在可视化算法的展示的层面上，而是在前人的基础上，设计与实现更为复杂的“交互式”功能[1]。例如，可视化算法交互式平台不仅可以让算法的用户看到对应的执行过程以及结果，还能引入多种常见算法，将这些算法纳入算法库中，方便用户在线查阅与学习，让用户将精力主要投入于“算法思维学习”而避免将过多时间耗费于语言语法，甚至能在线编译代码，让用户亲身感受执行过程。同时，可视化算法交互式平台更进一步地侧重于“交互”，用户可以自行调整代码来可视化自己的程序，也可以上传自己的代码纳入算法库中，这不仅仅可以让用户更快更深更准确地掌握对应的算法思维，也能更好地培养用户的代码能力。

基于上述讨论，文章确定了研究的主要内容有以下几个方面：（1）从便捷性使用的角度优化基础设计思想，使得系统最终能摆脱对环境的依赖，用户也不用考虑各种配置问题，

也不需要账号密码去登录系统，达到“开箱即用”的效果。（2）从设计一款高效的开发工具的角度出发，尽可能地实现轻量编辑器的主要功能，比如代码提示、语法高亮、缩进管理等功能。（3）从高效编译的角度出发，研究编译器的实现原理，参考主流编译器的设计思想，结合编译原理的知识，优化编译算法，在保证能正确编译执行代码的基础上优化性能，尽力实现高效编译。（4）从视觉效果的角度出发，参考主流可视化系统，先以排序算法大等经典算法为研究对象，尽量让最核心的可视化部分在视觉效果上能增强趣味性，并且在可视化渲染的兼顾可控性，考虑到算法的复杂性直接关系到渲染的复杂性，要在渲染时给用户控制权，比如暂停渲染、重置渲染。（5）从系统易用性的角度出发，不仅要囊括常见的算法，还要让用户能在系统中动态增加删除和修改算法，实现系统的动态扩展性。

1.4 论文的组织结构

第一部分是绪论，先介绍了算法可视化平台的研究背景与意义，再阐述国内外可视化平台的研究现状，最后表明本文研究的主要内容。

第二部分是系统分析，先从学生和老师这两种角色的角度分析了对算法可视化平台的具体需求，再从经济、社会和技术三个角度来分析可视化算法交互式平台的具体可行性。

第三部分是相关技术介绍，介绍了本文主要涉及到的技术工具以及设计模式，为之后阐述系统的整体设计与功能模块做铺垫。

第四部分是系统概要设计，先从技术架构的角度总结性地描述了整体的设计思想，再从功能模块的角度将系统分成了四个核心模块，最后分别阐述了这四个模块的核心功能以及设计思想，为之后阐述系统的详细设计细节与实现做铺垫。

第五部分是系统功能详细设计与实现，分别详细阐述了四个核心模块的流程设计、核心代码以及实现效果，最后再从系统的完整性与可扩展性的角度出发，分别阐述了系统算法库以及算法库的动态扩展的实现效果。

第六部分是总结与展望，先是对全文工作进行总结，之后再对文章中所存在的问题提出以后的研究和改进方向。

2 系统分析

2.1 需求分析

多年来，我们的教学工作一般都是写在黑板上，口头授课，PPT 演示，使师生形成了一定的思维定势。随着时代的发展和科学技术的进步，计算机辅助教学在教育教学改革中的地位越来越重要。我们可以利用计算机来帮助教师履行教学职能，教师利用计算机辅助

教学手段，利用计算机多媒体教学方法已经成为激发学生兴趣的一种新方法。例如，在教学中，教师可以使用计算机呈现教学计划、教学内容、记录学生的学习情况和控制学习过程。教师在教学中可以根据本学科的特点使用辅助软件，使整个课堂从原来抽象僵化的氛围变为生动活跃的氛围，充分发挥教师的领导作用和学生的主体性。

该系统是一个动态演示数据结构算法执行过程的辅助教学系统。它可以满足读者对算法输入数据和过程执行控制方式的不同需求。它可以在计算机屏幕上显示算法执行过程中数据逻辑结构或存储结构的变化，或递归算法执行过程中堆栈的变化。整个系统采用菜单驱动模式，每个菜单包含多个菜单项。每个菜单项对应一个操作或子菜单。在选择退出操作之前，系统始终处于选择菜单项或执行操作的状态。您可以选择在线编译和运行代码，或者进入可视化页面以可视化算法。

2.2 可行性分析

就经济可行性而言，系统在开发过程中使用的所有工具都是免费的开源工具，在保证高可靠性和良好性能的条件下，减少了资金支出。

在社会可行性方面，系统主要满足两类用户，即学生群体和教师群体，系统既对学生群体主要关心的算法执行过程进行了可视化渲染，又满足了教师群体主要关心的在线编译运行功能。

在技术可行性方面，系统采用的是用 C++实现分离编译模式，再用 C++写服务端，用 JavaScript 结合 Vue 与 LayUI 等作为前端技术框架开发。C++能精确控制内存使用，对 Latency 有着非常强大的控制能力，而且 C++语言很接近于底层，又具备面向对象的设计理念，现在互联网上走 TCP 协议的流量基本都是 C/C++的程序在控。C/C++组织源代码以及生成可执行文件的方式就是通过分离式编译模式实现的，我们在实际开发项目时，不太可能在一个头文件中放入所有的源程序，而是我们将不同的模块交给不同的开发工程师开发，最后，我们会将这些模块全部汇总成一个新的可执行程序，这在很大程度上可以极大地提高编译速度。Vue 是一门渐进式框架[12]，其响应式编程以及组件化特性非常契合系统的可视化流程[13]，而且 Vue 是单页应用，使页面局部刷新，不用每次跳转页面都要请求所有数据和 DOM[14]，这样大大加快了访问速度和提升用户体验。我们所使用到的第三方 UI 库 LayUI 是一个十分轻量的框架，它非常简单所以很容易掌握，并且 LayUI 有着简洁美观的特性，特别是在开发具有后端模式的项目时，它能将请求快速交由服务端并且也能快速取得响应。在算法可视化的分析过程中，为了实现算法的可视化，考虑了很多问题，总结出以下三个关键性技术：（1）用户图形界面设计：用户图形界面设计是软件设计的关键工

作之一，设计界面时要考虑到用户的使用，方便用户的使用，操作简单，采用动画技术，最大限度地调动用户的使用度。对于不同的算法程序采用与其相匹配的初始化设置界面。（2）图形化动态演示：图形化动态演示是将算法由抽象变具体，使学生的学习由枯燥变得生动。算法可视化软件中，颜色、声音、动画等多媒体元素的使用使界面变得漂亮，故算法可视化软件可以将多媒体元素合理使用，使软件的可视化达到最佳效果。（3）算法动态跟踪及演示：在算法的动态演示的过程中希望能很好的捕捉算法的动态运行过程。有的软件还需要对算法进行中断和单步执行来实现图形化演示程序的动态执行过程。这个问题的分析很难，需要对所使用的设计软件有深入的理解和对算法有深刻的认识。

3 相关技术介绍

3.1 LayUI

LayUI 是一款采用自身模块规范编写的前端 UI 框架，遵循原生 HTML/CSS/JS 的书写与组织形式[15]，门槛极低，拿来即用。该框架有个极大的好处就是定义了很多前后端交互的样式接口，比如分页表格，只需在前端配置好接口[16]，后端则按照定义好的接口规则返回数据，即可完成页面的展示，极大减少了后端人员的开发成本。它的体积很小，内部实现的功能却是非常多，它的组件丰盈，从核心代码到 API 的每一处细节都经过精心雕琢，非常适合界面的快速开发[17]。LayUI 属于轻量级框架，简单美观，适用于开发后端模式，它在响应服务端的页面上有非常好的效果[18]，而且 LayUI 是提供给后端开发人员最好的 UI 框架，尤其是对于基于 DOM 驱动以及不涉及到频繁交互的项目，LayUI 拥有非常好的性能[19]。

3.2 Cprase

Cprase 虚拟机是一个专用于项目解析运行代码的虚拟机。Cprase 使用 LR 词法分析进行 C 语言的语法解析，明确哪些东西需要 tracer 掉，在编译器中寻找 tracer，然后显示解析 tracer 并生成 ir，在虚拟机中记录下来，最后返回 tracer 步骤的 JSON 数据。

3.3 LR 分析法

LR 分析法也是一种自底向上的“归约移动”语法分析方法[20]。其实质是规范还原，具有以下特点：它有着广泛的应用，即可以通过 LR 分析程序识别大多数编程语言的语法结构；它的执行效率高，即虽然构造方法复杂，但执行效率高[21]；它的错误检查是准确的，即 LR Analyzer 可以及时发现语法错误并准确指出错误位置。在 LR(k)分析方法中，1 是指从左向右扫描输入字串，R 是指分析过程是最右边推导（规范化归约）的逆过程，k 是指

确定当前分析动作时要向前看的输入符号数。LR(0)结构简单，适用语法少。它是 SLR(1)和 LR(k)的基础，而 SLR(1)改善了 LR(0)的一些问题[22]。

LR 分析仪的逻辑结构如下：它在通用控制程序的控制下，从左到右扫描输入字符串，根据分析堆栈和输入符号检查分析表，确定分析动作，它的分析表是 LR 分析仪的核心，它是根据语法构造的，总共包括两部分：动作表和状态转换表[23]。通用控制程序再根据分析表确定分析动作，之后它的分析堆栈由语法符号堆栈 x[i]和相应的状态堆栈 s[i]组成。LR Analyzer 通过判断堆栈的顶部元素和当前输入符号并查找分析表来确定下一个分析操作。

规范化归约的关键是找到归约的句柄。LR 方法分析已“移入”、“缩减”的符号字符串和将被读入的符号字符串，以确定是否存在可以缩减的句柄。状态是到目前为止整个分析过程的记录，以及要扫描的输入字符串的前景（即预测）[24]。例如，S0 状态（开始状态）表示当前堆栈中只有“#”，要输入的符号是句子开始符号，Sn 状态（接受状态）表示当前堆栈已缩减为开始符号，要输入的符号应该是“#”，状态是由语法构造的分析表获得的[25]。

LR 分析法的流程图如图 1 所示。

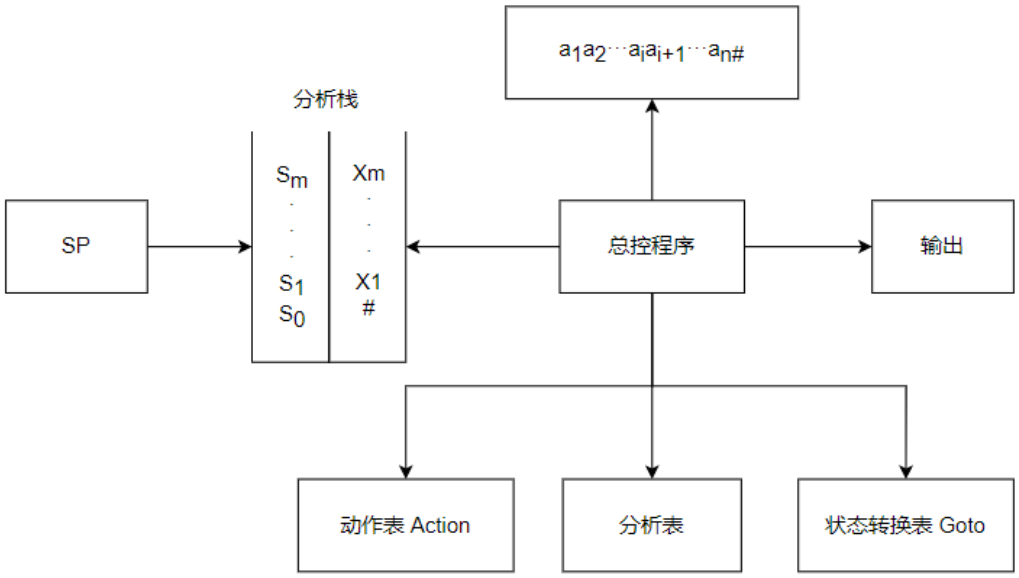

图 1 LR 分析法流程图

3.4 Libevent

Libevent 是开源社区中的一个高性能 I/O 框架库[26]。该库支持跨平台、统一事件源，并且是线程安全的[27]。最重要的一点是，该库使用的是 Reactor 模式，使用库后，服务器端的性能响应迅速，不需要被单独的同步事件阻止，写作相对简单。它可以最大限度地避

免复杂的多线程同步问题，避免多线程/进程的切换开销，具有可重用性，Reactor 框架本身独立于特定的事件处理逻辑，具有很高的可重用性。Libevent 是一个用 C 语言编写的轻量级开源高性能事件通知库。它有着事件驱动、高性能、轻量级、专注于网络的特点，又没有 ACE 那么笨重，它的源代码非常精炼，易于阅读，它也支持跨平台使用，支持 Windows、Linux、*BSD 和 MacOS[28]，也支持多种 I/O 复用技术，比如 epoll、poll、dev/poll、select 和 kqueue，它也支持 I/O、定时器和信号等事件。Libevent 注册了事件优先级，被广泛用作底层网络库，比如呕吐物，尼龙，网络聊天。Libevent 是一个事件通知库[29]，它在内部使用系统调用管理事件机制，如 select、epoll、kqueue 和 iocp，著名的分布式缓存软件 Memcached 也是基于 Libevent 开发的[30]。

3.5 RapidJSON

RapidJSON 是一个只有头文件的 C++开源库，它是十分高效的解析器和生成器，用于处理 JSON 开源代码，可以解析和转换网页请求的 JSON 数据。它可以跨平台使用，它支持 Windows、Linux、MacOS、IOS 和 Android。它同时支持 SAX 和 DOM 风格的 API。它是独立的，不依赖 boost 和其他外部库，它对记忆很友好。它可以用于高效、快速的 JSON 解析，以提高服务器性能。它很小，很完整。它支持 SAX 和 DOM 风格的 API，SAX 解析器只有大约 500 行代码。与此同时，它的加载速度很快，性能可与 strlen()媲美。它可以支持 SSE2/SSE4.2 加速，使用模板和内联函数来降低函数调用的成本。它非常独立，它不依赖 boost 等外部库，甚至不依赖 STL。它对内存友好，在大多数 32/64 位机器上，每个 JSON 值只会占用 16 个字节（字符串除外）。它预设为使用快速内存分配器，以便分析仪可以紧凑地分配内存。每个 JSON 值都存储为一个值类，而 Document 类代表整个 DOM，它存储 DOM 树的根值。它的所有公开类型和函数都位于它命名空间中。它对 Unicode 很友好。它支持 UTF-8、UTF-16 和 UTF-32（大端序列/小端序列），并在内部支持这些代码的检测、验证和转码。例如，在将 UTF-8 文件分析为 DOM（文档对象模型）时，它可以将 UTF-8 文件中的 JSON 字符串转换为 UTF-16。

3.6 分离式编译模式

分离编译模式源于 C 语言，在 C++语言中继续沿用。简单地说，分离编译模式是指：一个程序由若干个源文件共同实现，而每个源文件单独编译生成目标文件，最后将所有目标文件连接起来形成单一的可执行文件的过程。每个.cpp 源文件经过预处理，最后会展开它所包含的.h 文件的代码，再经过编译器的编译汇编等过程，将该.cpp 文件转变为.obj 文件，这是此文件已经变为二进制文件，本身包含的就是二进制代码。这时，该文件还不一

定能够执行，因为并不保证其中一定有 main 函数，或者该源文件中的函数可能引用了另一个源文件中定义的某个变量或者函数调用，又或者在程序中可能调用了某个库文件中的函数等等。这些都要通过链接器将该项目中的所有目标文件连接成一个单一的可执行文件来解决。就最常用的 Visual Studio 来说，对于每个 cpp 都是独立编译之后再进行链接的，而为了保证一个 cpp 能调用另一个 cpp 而不会导致另一个被编译两次（include 另一个文件会在预处理时展开并参与编译），一般都会把类和函数的声明写在.h 文件以及.cpp 的 include 之中。所以一般的类和函数都会在编译时确定它的声明但没有定义，会留到链接时与其他文件中的定义进行链接。

分离编译模式是 C/C++组织源代码和生成可执行文件的方式。在实际开发大型项目的时候，不可能把所有的源程序都放在一个头文件中，而是分别由不同的程序员开发不同的模块，再将这些模块汇总成为最终的可执行程序。这里就涉及到不同的模块（源文件）定义的函数和变量之间的相互调用问题。C/C++语言所采用的方法是：只要给出函数原型（或外部变量声明），就可以在源文件中使用该函数（或变量）。每个源文件都是独立的编译单元，在当前源文件中使用但未在此定义的变量或者函数，就假设在其他的源文件中定义好了。每个源文件生成独立的目标文件（obj 文件），然后通过连接（Linking）将目标文件组成最终的可执行文件。程序编译的简要过程包括预处理（Preprocessing）、编译（Compilation）、汇编（Assembly）和连接（Linking），每个源文件都是独立的编译单元，即编译是相对于每个 cpp 文件而言的。分离编译模式如图 2 所示。

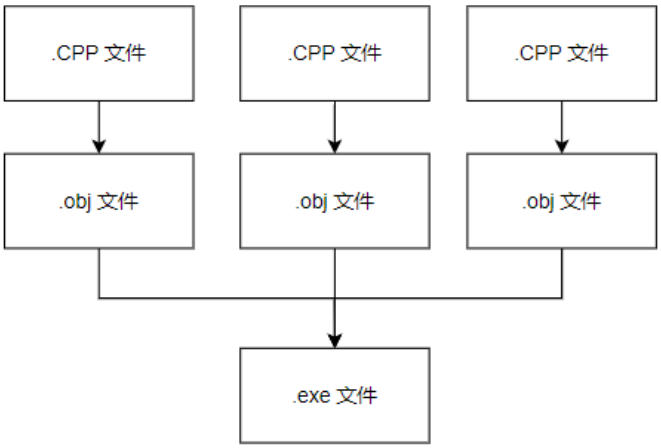

图 2 分离式编译模式图

4 系统概要设计

4.1 总体架构设计

系统整体上采用的是 B/S 架构，这使得业务扩展非常简单方便，可以直接通过添加网页来添加服务器功能[31]，系统从网页中获取用户要编译的代码或者可视化的算法，再发送给服务器，服务器处理之后返回 JSON 数据，前端收到服务器的反馈之后解析 JSON 数据展示结果，展示结果使用的是 Chart.js 来渲染出图表[32]，Chart.js 是一个非常简单而且非常灵活的 JavaScript 图表工具，它根据浏览器窗口调整图表的大小，并为不同的窗口大小提供完美的缩放粒度[33]。

后端采用 C++的第三方库 Libevent 进行请求的轮询等待和处理，该库使用的是 Reactor 模式，用库后，服务器端性能会快速响应，而不会被单个同步事件阻止。用户在网页中提交代码或者输入可视化的代码，之后直接提交给服务器，服务器接收到浏览器发出的请求之后，Libevent 会提供对应的请求处理函数去解析请求，最后会选择对应的 API 处理函数去处理用户的请求。

在算法的可视化解析方面采用的是微型虚拟机 Cprase，Cprase 虚拟机是专用于项目解析运行代码的微型虚拟机，此微型虚拟机中其中含有一个 GUI 类，专用来进行代码可视化的解析。之后我们采用 RapidJSON 来解析和转换 JSON 数据，我们可以用它来进行高效、快速的 JSON 解析，提高服务器性能。整个系统最后可以生成一个可执行文件，启动此文件就可以在本地开启服务。系统的总体架构如图 3 所示。

4.2 功能模块设计

系统主要包括 4 个模块：代码编译模块、请求处理模块、可视化解析模块、动态渲染模块。

代码编译模块的核心功能是在线编译。请求处理模块对请求解析完成之后，选择对应的 API 处理函数，将用户要编译的代码的路径告诉代码编译模块，代码编译模块找到对应的代码，将其加载到服务器内存中，使用 Cprase 虚拟机对其进行语法解析和使用 programing 函数对解析后的代码执行，并将处理后的结果返回给服务器，服务器进行后续的处理，如果是编译请求则直接返回运行结果，如果是可视化请求，就交给可视化解析模块完成后续的处理。

请求处理模块的核心功能是处理请求。用户在前端提交代码，或者选择要可视化的算法，之后直接提交给服务器。服务器在接收到浏览器发出的请求之后，Libevent 会提供对应的请求处理函数，该函数会对请求进行解析，最后会提取出其中用户的需求，然后选择

对应的 API 处理函数，处理用户的请求。等待其他模块将用户的请求处理完之后，将处理完成后的数据添加 HTTP 报头生成一个 HTTP 报文，返回给前端，交由前端进行后续处理。

系统的总体架构图如图 3 所示。

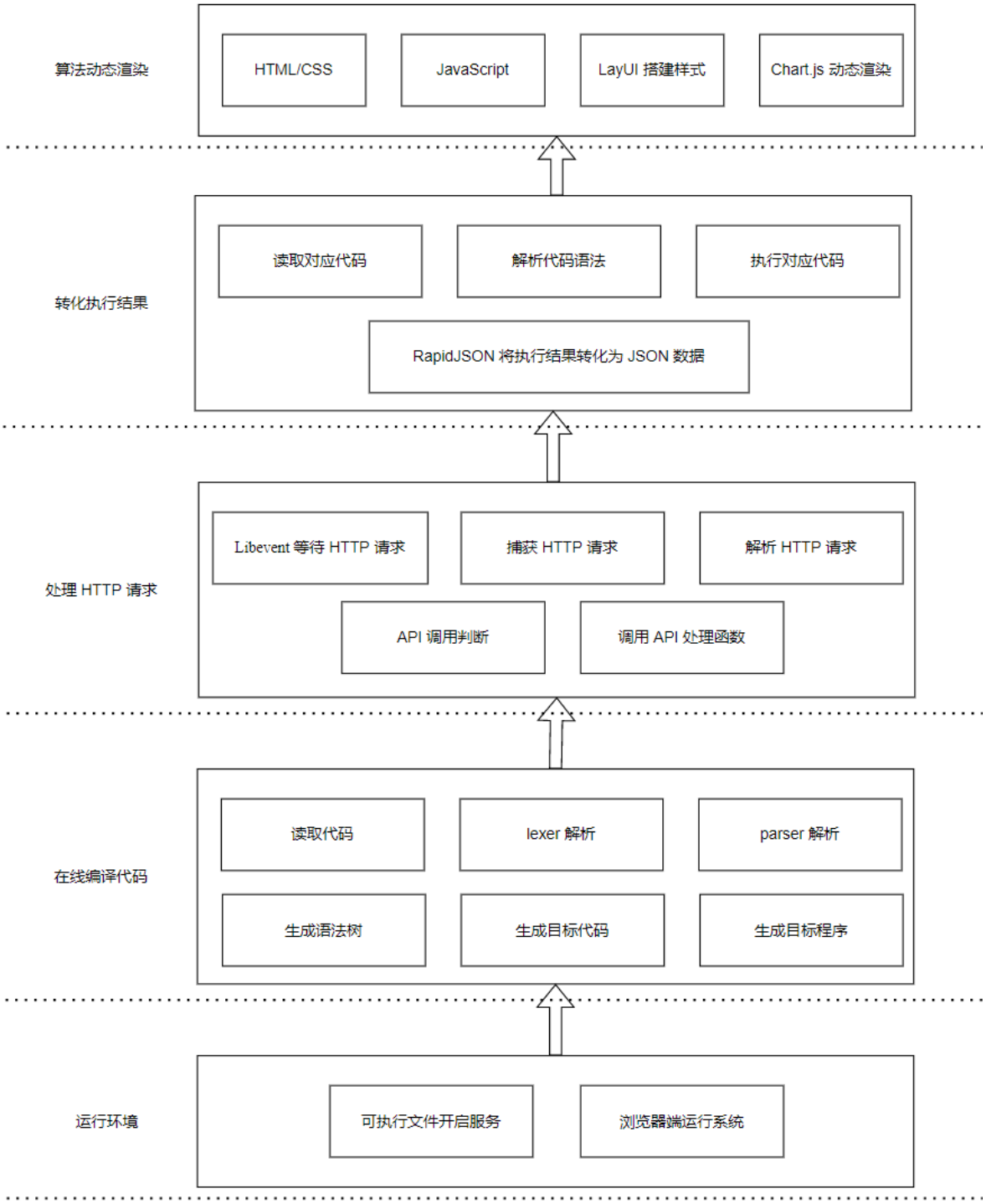

图 3 系统总体架构图

可视化解析模块的核心功能是将执行结果转化为 JSON 数据。等待代码编译模块完成后，如果是可视化请求，则会进入到模块进行可视化的解析，模块执行时，先进入到 Cprase 虚拟机中查看 trace_recods 表，如果表为空则给输出添加一个错误标志，表示没有要可视化的数据，如果不为空，则创建一个 GUI 类，调用 GUI 类中的 draw 函数对执行完成后的代码进行可视化，再调用 tracer_json 对可视化后的数据进行格式转换，转换为 JSON 格式，再交由服务器处理。

动态渲染模块的核心功能是对获取到的 JSON 数据进行持续渲染。获取到 JSON 格式的数据之后，由 Chart.js 将 JSON 数据渲染成图表等形式的可视化样式，Chart.js 本身是静态渲染的，但是由于会一直获取 JSON 数据，所以会及时更新待渲染的 JSON 数据源，所以最后呈现出来的是动态渲染的过程。

系统的功能模块图如图 4 所示。

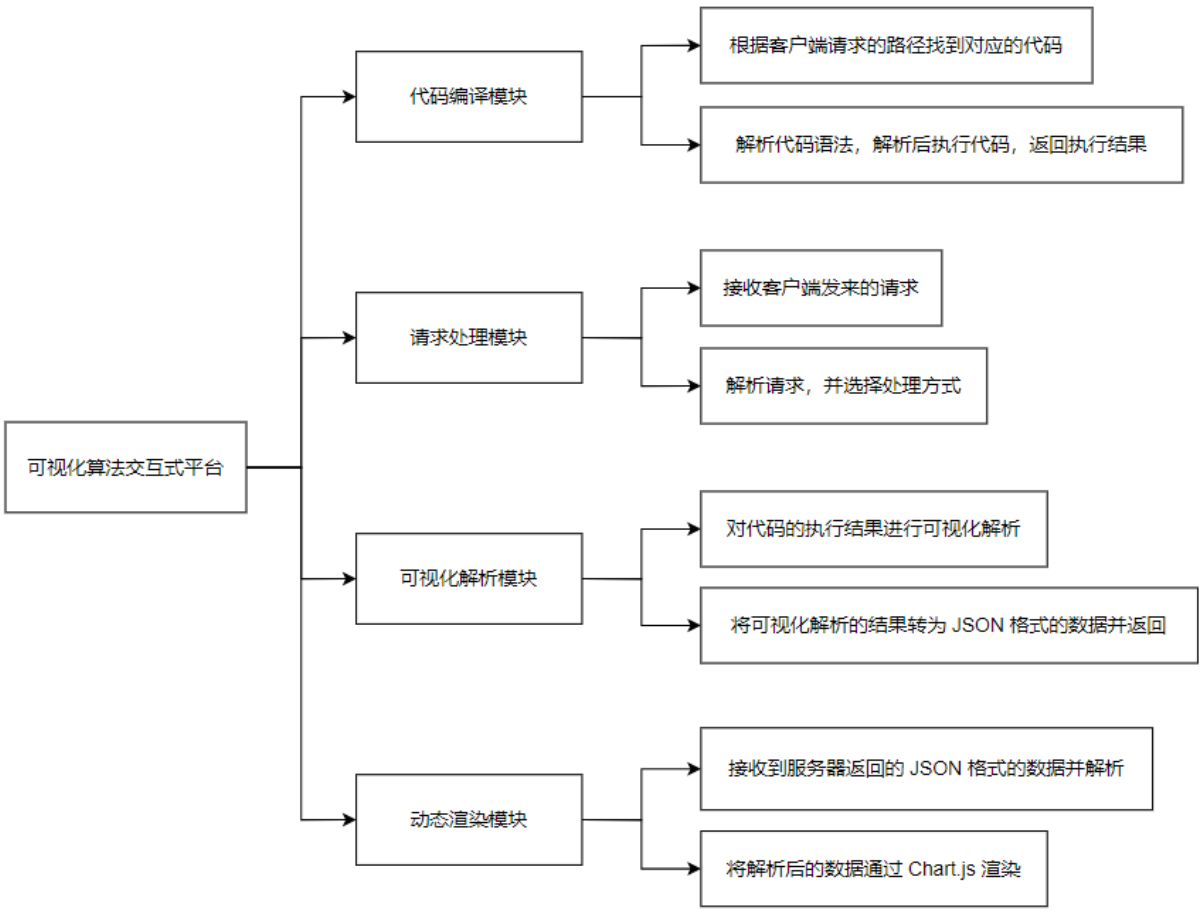

图 4 系统功能模块图

5 系统功能详细设计与实现

5.1 代码编译模块

5.1.1 流程设计

编译器分为前后两端，编译器的主要工作流程：前端主要对代码进行分析，对代码进行语义分析、语法分析、词法分析等操作，最后生成中间代码，并将中间代码传输到后端。后端的主要工作是优化中间代码，然后生成目标代码，之后再优化目标代码，最后生成目标程序。

编译器前端设计设计采用主要 LR 文法分析，可以根据笔者设立的 LR 规则去生成 LR 状态表，然后根据状态表进行文法解析，要实现 LR 文法分析，需要实现生成 AST（抽象语法树）功能则还需要实现两个模块 lexer 和 parser 模块，lexer 模块完成词法分析，返回各类终结式（token 流），并且将终结式写入到对应文件中，parser 完成语法分析，从 lexer 生成的文件中读取终结式作为输入产生式，通过 LALR 表来分析输入产生式，分析完成后成功生成 AST。

通过以上两个模块，即可生成 AST，AST 是源代码抽象语法结构的树表示，树上的每个节点代表源代码中的一个结构。因此，它是抽象的，因为抽象语法树并不代表真实语法的每个细节。例如，嵌套括号隐藏在树结构中，并且不会以节点的形式显示。

代码编译模块的流程图如图 5 所示。

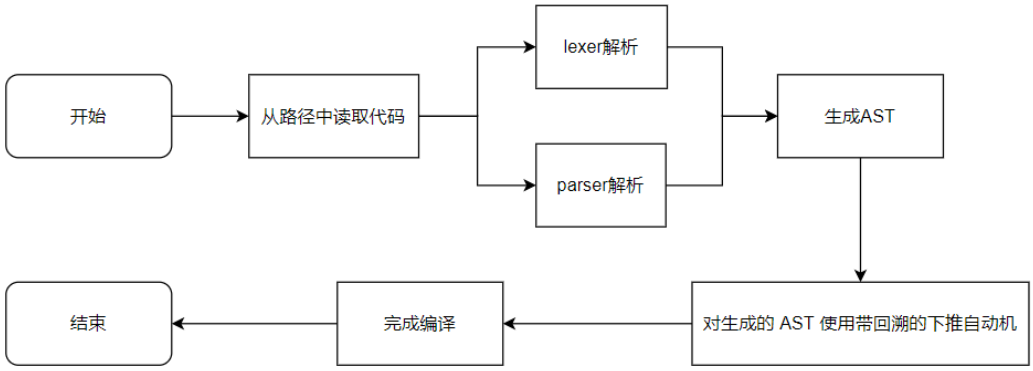

图 5 代码编译模块流程图

通过上述步骤生成了 AST 之后，再使用带回溯的下推自动机，完成编译，最后将 C 语言代码加载到内存之后，交给编译器时，编译器执行上述步骤，解析 C 语言语法，进行执行，即可完成 C 语言的编译，然后再执行编译后的文件即的到执行结果。我们将源代码

转化为 AST 后，可以对 AST 做很多的操作，这些操作实现了各种各样形形色色的功能。通过以上功能的实现，并且加上带回溯的下推自动机的实现，即可完成 C 语言的解析。

编译器的编译流程图如图 6 所示。

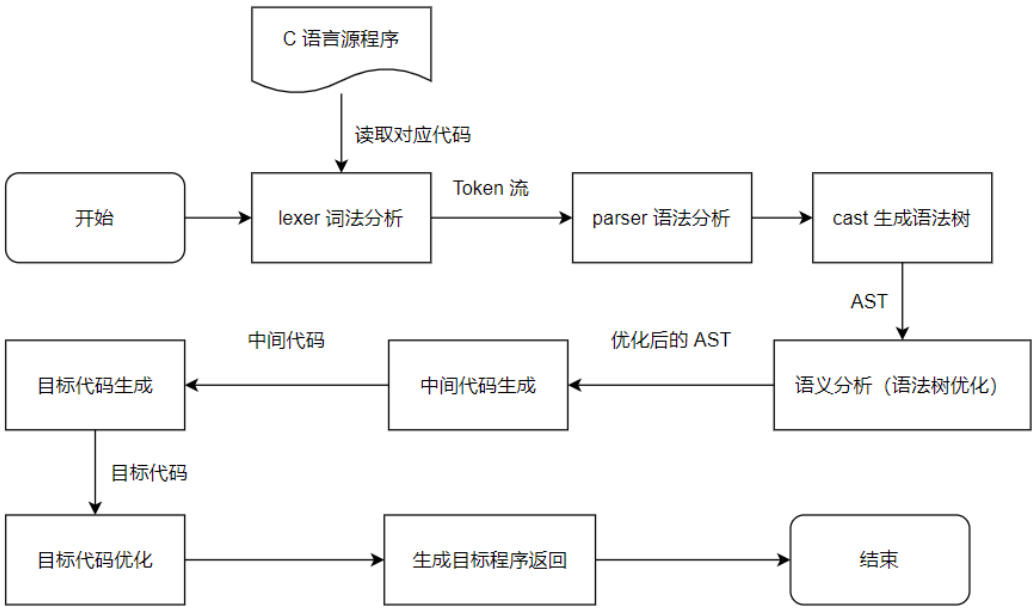

图 6 编译器编译流程图

parser 语法分析器是在 lexer 词法分析的基础之后，对其分割的 token 流，分析语法，分析源程序的语法结构，并将线性令牌流转换为树结构（语法分析树）。解析器的设计采用了与上下文本无关的语法。与上下文本无关的语法包括终止符集、非终止符集、产生式集，以及一个起始符号，由语法推导产生。因此一个语法可以代表一个句子集合，也是一个语言。通过这个语法，我们可以判定句子是否符合语法结构，再进行后面的操作，其中 LL(1)文法就是进行这样的判断，如果符合就构建预测分析表，分析这个句子是如何从起始符号生成的，并在分析过程中生成成语树。

根据以上两个步骤产生的结果，使用 cast 模块，生成 AST，该模块使用的自顶向下分析，使用带回溯的下推自动机，过滤掉了多余的步骤，提高效率，在回溯失败的时候，lexer 词法分析器不会从前面重新解析，而是缓存已解析的节点，并且产生的多余 AST（不包括词法 AST 节点而是集合节点）将会被删除，一次也会将多个分支合并为一个回溯结构。

接下来就是编译器的后端了，后端的主要工作是，将前端进行一系列解析过程后生成的 AST 进行优化，然后对优化后的 AST 进行语义分析，生成中间代码，再对生成的中间

代码进行优化，然后使用 NASM 将中间代码生成目标代码（x86 汇编指令），最后使用 NASM 的连接器 LD 将其连接成为可执行文件返回。

语义分析模块主要采用递归下降分析器的设计。递归下降分析使用函数之间的递归调用来模拟语法树的自上而下的构造过程。递归下降分析实现了以下两个功能：计算算术表达式的值和计算中间代码的四元式。

parser 语法分析的流程图如图 7 所示。

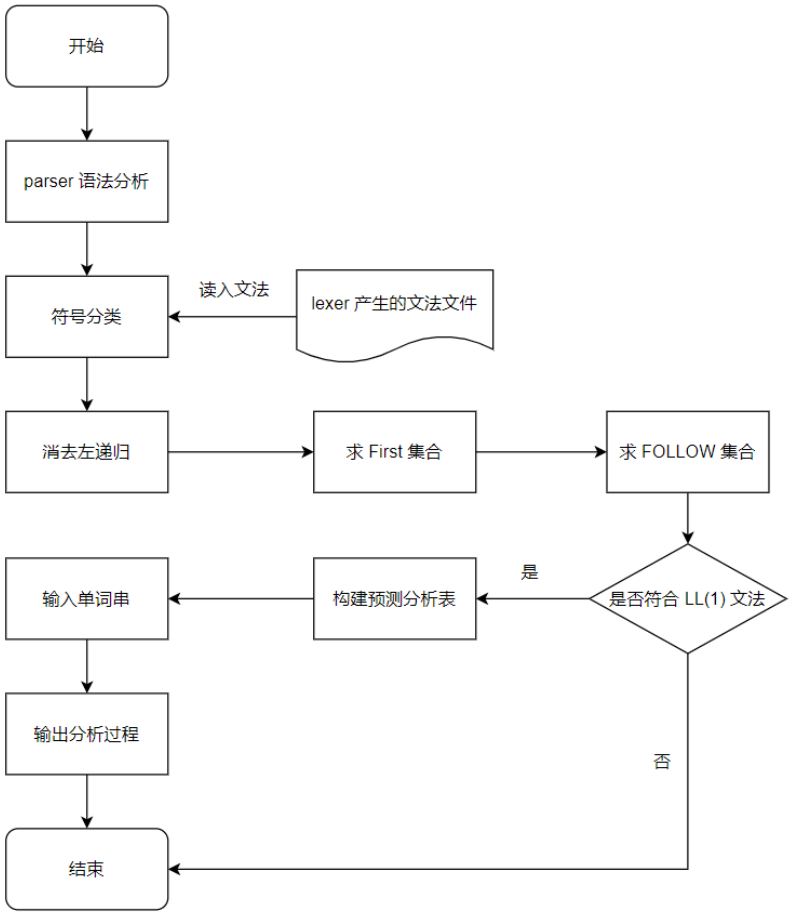

图 7 perse 语法分析流程图

其主要设计思想是，为每个非终结符构造一个函数过程，为每个继承属性设置一个形式参数，函数的返回值是其综合属性。对于每个产品，按照从左到右的顺序，对单词符号执行以下工作：对于具有综合属性的终止符，将它们存储在设置的变量中，生成一个调用，然后读取下一个输入。对于非终止符，使用函数调用生成有值语句。对于语义动作，将动作复制到分析器中，并用表示该属性的变量替换相应属性的引用。

lexer 词法分析阶段：在这个阶段，编译器从左到右扫描要分析的源文件，并将源文件

中的字符流划分为一个单词。这个词被称为 token，它是源文件中不能进一步分割的字符串。程序语言中的标记是有限的，通常包括常量、运算符、分隔符、保留字、标识符等，在设计 token 时，使用了一个枚举（TokenType）和一个结构体（struct _Token），来标识一个唯一的 token，lexer 词法分析器每扫到一个完整的 token，就代表该状态是最终完成的状态(终态)，就立即新建一个 TokenRecord，将 token 记录在其中，以便 parser 进行语法分析时访问。当语法分析完成后，TokenRecord 就变成了一个连续的 TokenStream。在扫描时，每轮扫描根据第一个字符确定字符属于哪种类型的令牌，然后使用不同的策略扫描完整的令牌，然后执行下一轮扫描。lexer 词法将 token 分为七种判定类型，单字运算符、双字运算符、关键词和标识符、整数常量、字符串常量、空格、注释。

语义分析流程图如图 8 所示。

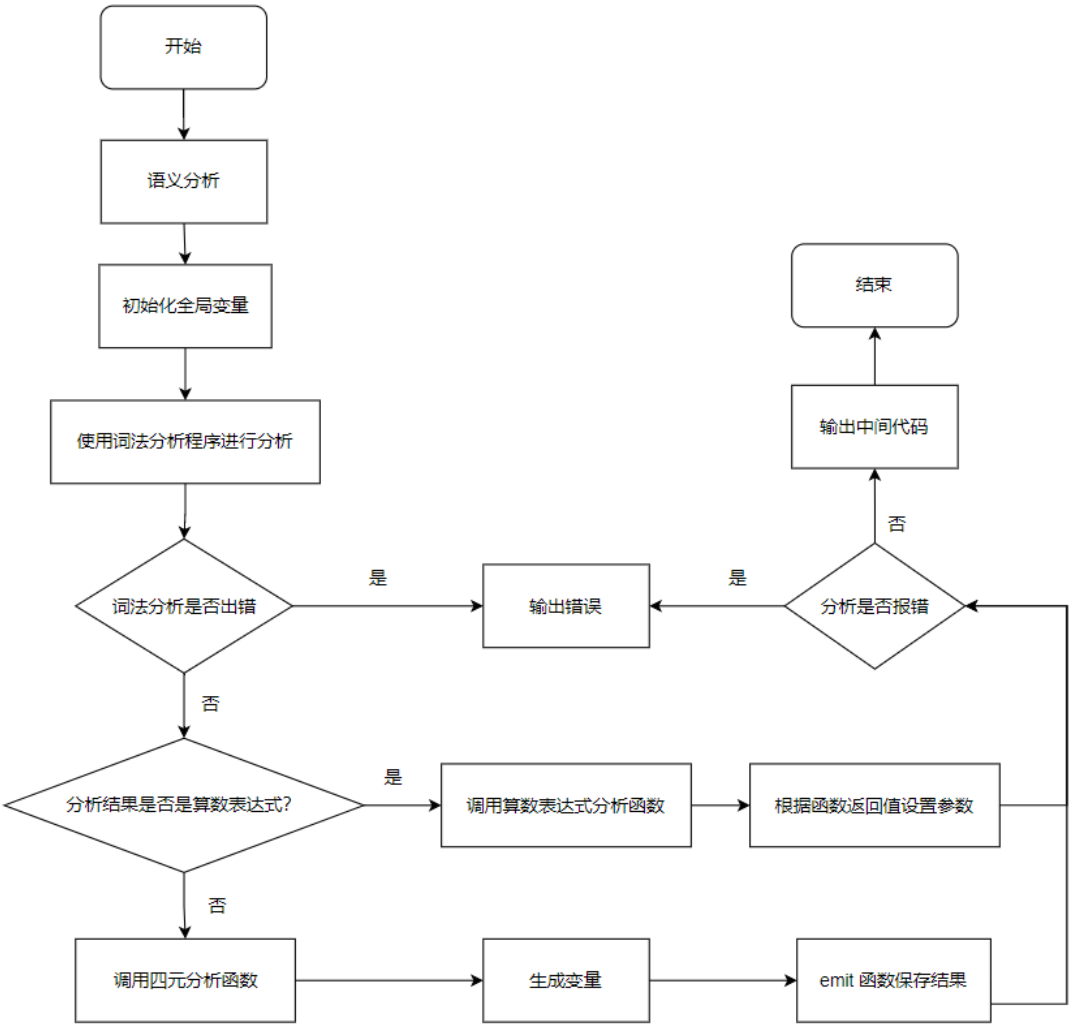

图 8 语义分析流程图

lexer 语法分析流程图如图 9 所示。

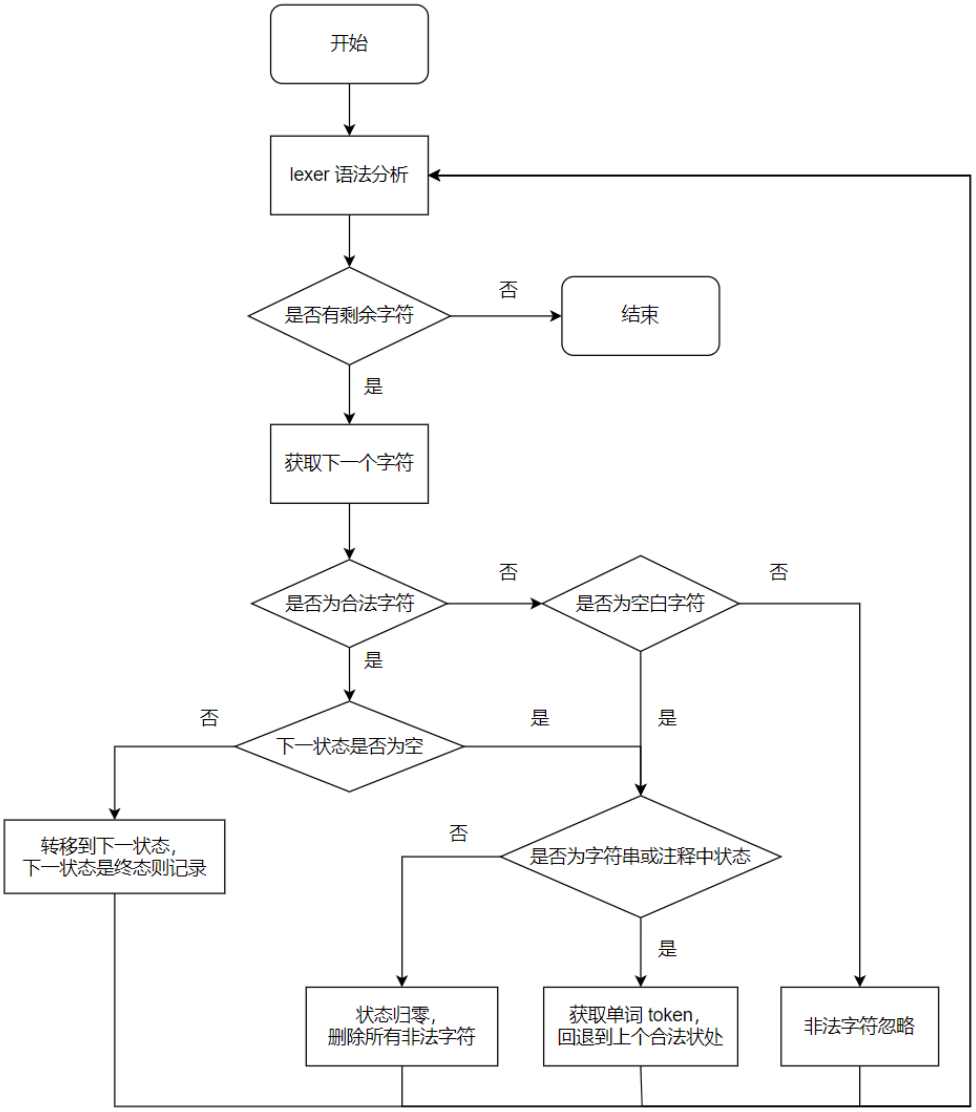

图 9 lexer 语法分析流程图

5.1.2 核心代码

该模块中解析器的功能是检查语法并构造由输入单词组成的数据结构。通常，它是一种层次化的数据结构，如语法分析树和抽象语法树，它的核心代码如下：

```
ast_node* cparser::parse(const string_t& str, csemantic* s) {
    semantic = s;
    lexer = std::make_unique<clexer>(str);
    ast = std::make_unique<cast>();
```

```
    lexer->reset();                          // 清空词法分析结果
    ast->reset();                            // 清空 AST
    if (unit.get_pda().empty()) gen();       // 产生式
    program();                               // 语法分析（递归下降）
    return ast->get_root();
}
```

lexer 的作用是将原始字符串转换为有意义的标记的过程，当 lexer 词法分析器扫到一个完整的 token，就代表该状态是终态，就立即新建一个 TokenRecord，以便 parser 进行语法分析时访问。它的核心代码如下：

```
lexer_t clexer::record_error(error_t error, uint skip) {
    err_record_t err{};
    err.line = line;                         // 起始行
    err.column = column;                     // 起始列
    err.start_idx = index;                   // 文本起始位置
    err.end_idx = index + skip;              // 文本结束位置
    err.err = error;                         // 错误类型
    err.str = str.substr(err.start_idx, err.end_idx - err.start_idx);     // 错误字符
    records.push_back(err);
    bags._error = error;
    move(skip);                              // 略过错误文本
    return l_error;
}
```

5.1.3 实现效果

在打开可执行文件之后，即表示开启了后端服务，此时就可以在浏览器输入 8868 端口使用系统，此端口会默认进入系统的首页，也就是在线编译的页面。

我们可以在左边的代码框中输入代码，在线编译功能与市面上的轻量编译器类似，有许多功能都已经实现，如图 11 所示，我们在左边的代码框中输入代码，示例代码是用递归的方法输出斐波那契数列的第 20 个数字，点击“发送”，则输出结果就打印到右边的显示框中。

在线编译首页如图 10 所示。

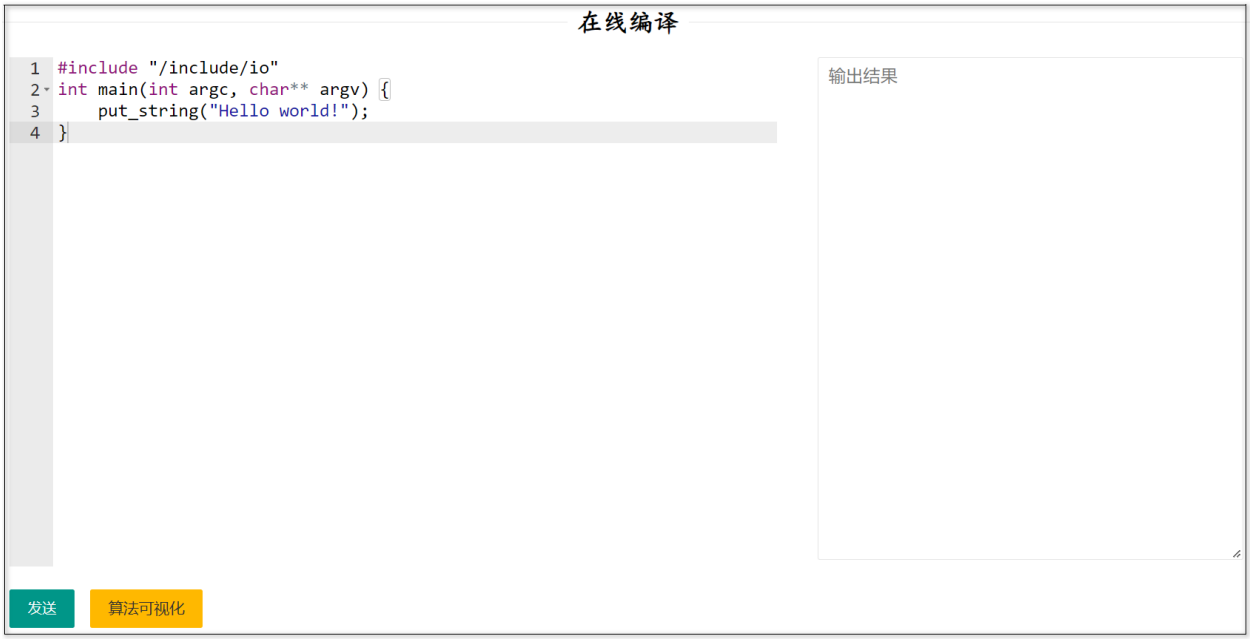

图 10 在线编译首页图

输出斐波那契数列的第 20 个数字结果如图 11 所示。

图 11 输出斐波那契数列第 20 个数字结果图

在左边的“代码框”中输入代码，代码可高亮显示，有实时的语法提示，语法提示框会实时地在下方显示，每一个提示后面都跟着灰色的类型说明文字，local 指的是保留字，keyword 指的是关键字，snippet 指的是流。

代码高亮以及实时提示如图 12 所示。

图 12 代码高亮与语法提示效果图

系统还可以对花括号所包含的代码进行压缩显示，未压缩的花括号内代码内容的样式如图 13 所示。压缩花括号内代码内容的样式如图 14 所示。

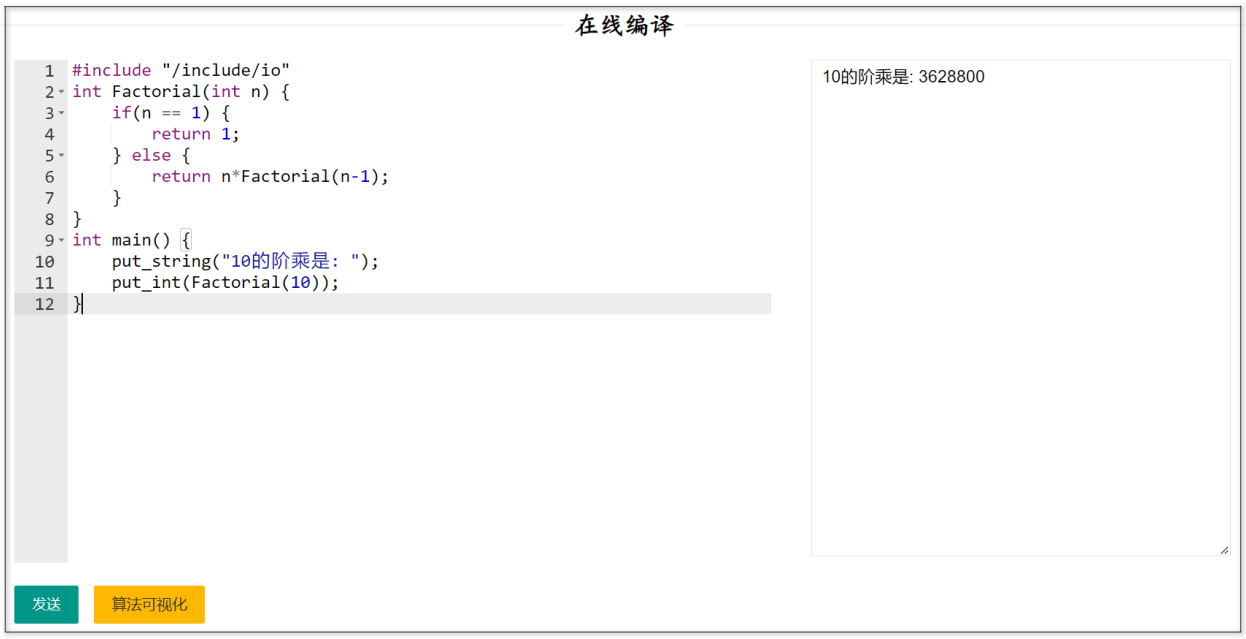

图 13 未压缩花括号内代码内容的样式图

图 14 压缩了花括号内代码内容的样式图

点击“发送”按钮即可在“输出结果”部分显示出结果，编译输出 “Hello World!” 的示例结果如图 15 所示。

图 15 在线编译成功的示例图

如果有语法错误，则在线编译的结果会将错误信息打印到“输出结果”框中，如图 16 所示，在左边的代码框中第 3 行代码最后没有加分号，这种写法在 C++的语法中是错误的，

所以右边的输出框中报错了。有语法错误的编译结果如图 16 所示。

图 16 在线编译语法报错的示例图

5.2 请求处理模块

5.2.1 流程设计

请求处理模块主要是用 C++设计的 Http 服务器，Http 服务器提供两个 API，读取 C 文件、返回 tracer 步骤 JSON，这是服务器的主要功能。

服务器处理请求的实现分为两部分捕获请求和处理请求，捕获请求 generic_handler，处理请求分为 handle_API 和 handle_static、handle_API 处理 Post 请求，handle_static 处理 Get 请求，捕获请求的主要实现是通过 Libevent 库来实现 Http 协议的解析和响应，解析了 HTTP 请求后对其进行判断，是 Get 还是 Post 请求，再来细化后面的操作。

如果是 Get 请求就返回静态网页，如果是 Post 请求就判断请求中 API 的类型，虽然只提供了两个 API 但是有三个 API 类型，分别为：API_compile、API_visualize、API_reload。

判断后如果 API 为 compile 那么服务器就从 Http 请求中拿出里面的 JSON 数据，使用 RapidJSON 库对 JSON 数据进行解析，解析出其中的代码后调用 C 语言编译器进行编译执行，将执行的结果再通过 RapidJSON 库中的函数转换成 JSON 格式的数据之后返回给前端。

请求处理模块流程图如图 17 所示。

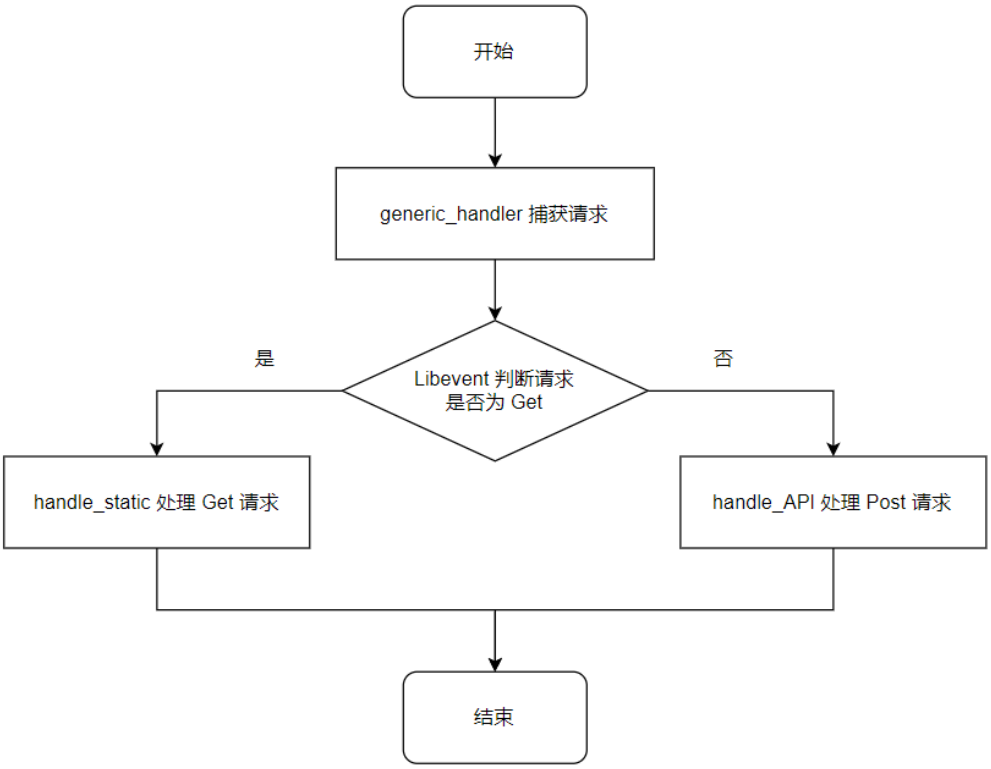

图 17 请求处理模块流程图

处理 Get 请求的流程图如图 18 所示。

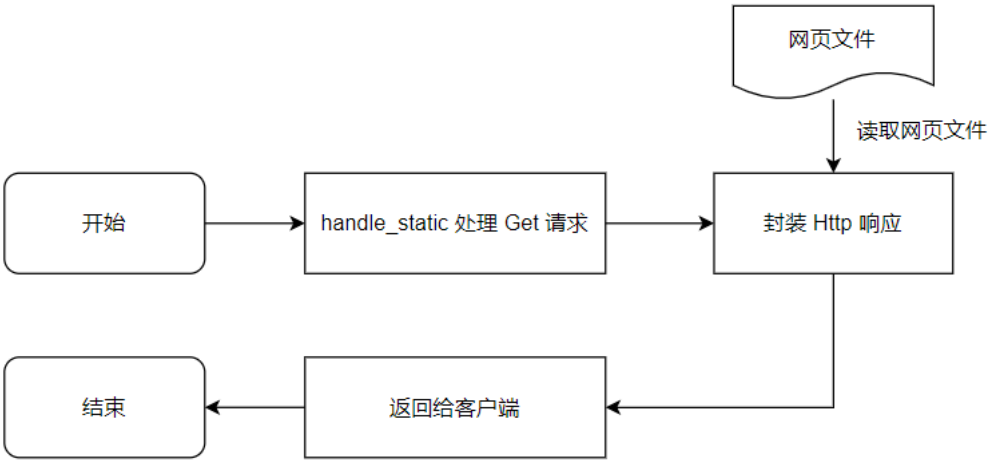

图 18 处理 Get 请求流程图

处理 Post 请求的流程图如图 19 所示。

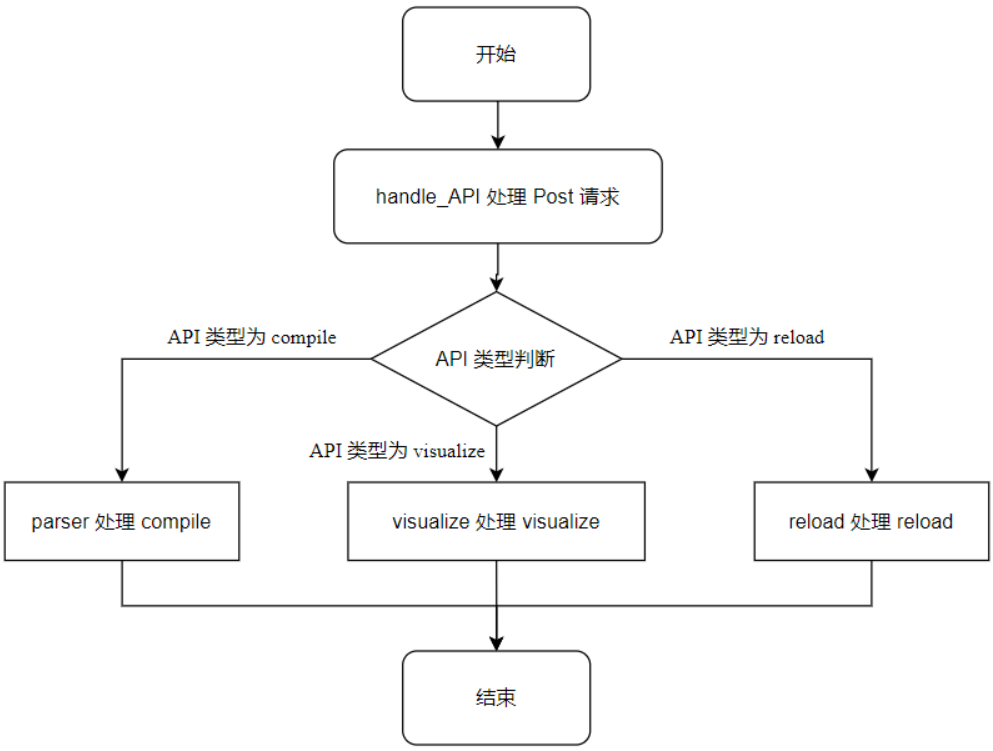

图 19 处理 Post 请求流程图

处理 compile 请求的流程图如图 20 所示。

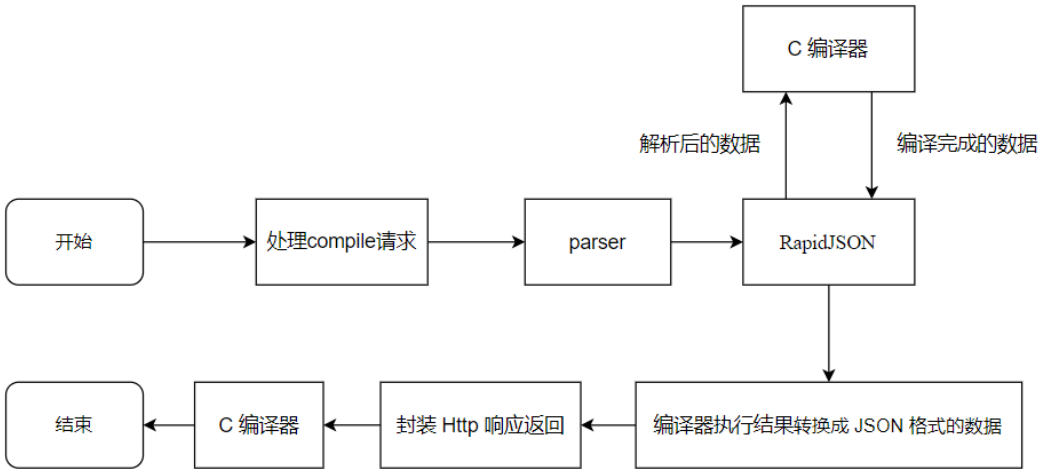

图 20 处理 compile 请求流程图

如果 API 是 visualize 的请求，就先对请求中的 JSON 数据进行解析，将解析之后根据 JSON 中提供的数据，找到服务器端中对应的代码，再将代码加载到服务器中，然后把代码交给 C 语言编译器执行，再将执行的结果使用 C 语言编译器中的 GUI 模块，进行可视

化将其转换为 Chart.js 能够解析的 JSON 数据格式，最后返回给前端。

处理 visualize 请求的流程图如图 21 所示。

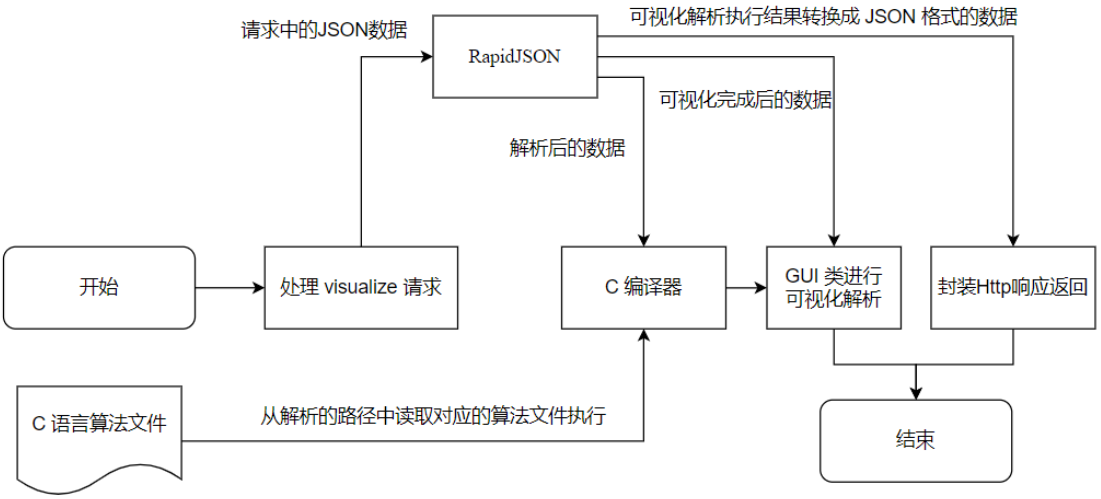

图 21 处理 visualize 请求流程图

如果 API 是 reload，那么代表这是一个刷新请求，则直接从 web 根目录加载页面文件。处理 reload 请求的流程图如图 22 所示。

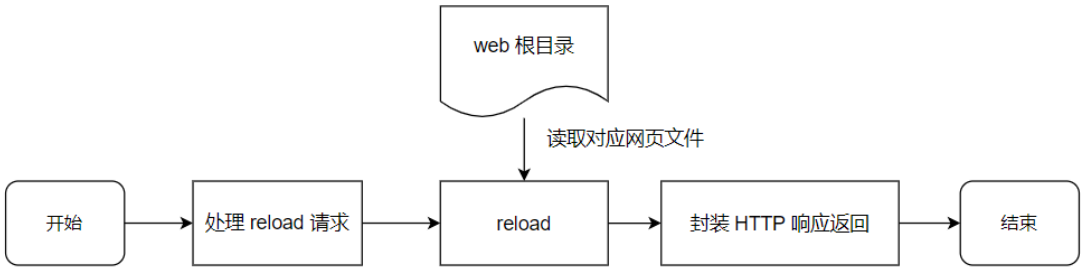

图 22 处理 reload 请求流程图

5.2.2 核心代码

判断完成后就对请求进行处理，处理时调用两个处理函数，针对不同的情况进行处理，其中 handle_static 的主要逻辑是，先依据请求中的 URL 找到文件，然后对 Http 的报头进行修改，将对应的文件放在主体中，最后返回，然后是处理 API 的逻辑，在 generic_handler 中，已经对 API 进行了判断，判断了是什么 API，所以 handle_API 函数，是用来处理特定的 API 的，要将 API 作为参数传入到 handle_API 中，函数内部再根据对应的 API 处理请

求，核心代码如下：

```
if (api == API_reload) {                    //判断请求、请求为 reload
    reload();
    return HTTP_OK;                         //重新加载页面返回 200 OK 状态码
} Document d;
if (d.Parse(data).HasParseError()) {        //判断接收到的数据是否成功解析
    evbuffer_add_printf(buf，  R"(Parse Error(offset %u): %s)")
    return HTTP_BADREQUEST;                 //解析失败返回请求错误、无法解析
} auto text = d.FindMember("text");         //从数据中找到解析后要访问的文件的路径
if (text != d.MemberEnd()) {                //判断是否找到
    Value& s = d["text"];
    if (s.IsString()) {
        auto str = std::string(s.GetString());
        decltype(str) out;
        parser_ret ret = P_ERROR;           //判断 JSON 解析是否成功
        if (api == API_compile)
            ret = parser(str，  out);       //编译代码
        else if (api == API_visualize)
            ret = visualize(str，  out);    //可视化
    } else
        evbuffer_add_printf(buf，  "{Need string}");
} else
    evbuffer_add_printf(buf，  "{Need text}");      //向缓冲区中写入返回的内容
return HTTP_BADREQUEST;                     //如果没有找到就返回请求错误
```

5.2.3 实现效果

使用系统前必须点击可执行文件来开启后端服务，所有的后端服务都由请求处理模块处理，所有的反馈信息都会打印到 CMD 窗口中，开启后端服务后，进入系统首页时的请求处理信息如图 23 所示。

图 23 加载首页时的请求处理信息图

在此基础上，我们点击首页中的“发送”，即我们在线编译代码框中的代码，此时浏览器会发出相应的 Post 请求，最终会在浏览器的输出框中中输出结果，处理 Post 请求的信息会打印在 CMD 中，如图 24 所示，CMD 窗口中显示处理了一条 Post 请求信息，并在旁边打印出了发出请求路径的是/api/compile，以及相应的时间是 21ms。

点击“发送”之后，处理 Post 请求来输出“Hello world”的打印信息如图 24 所示。

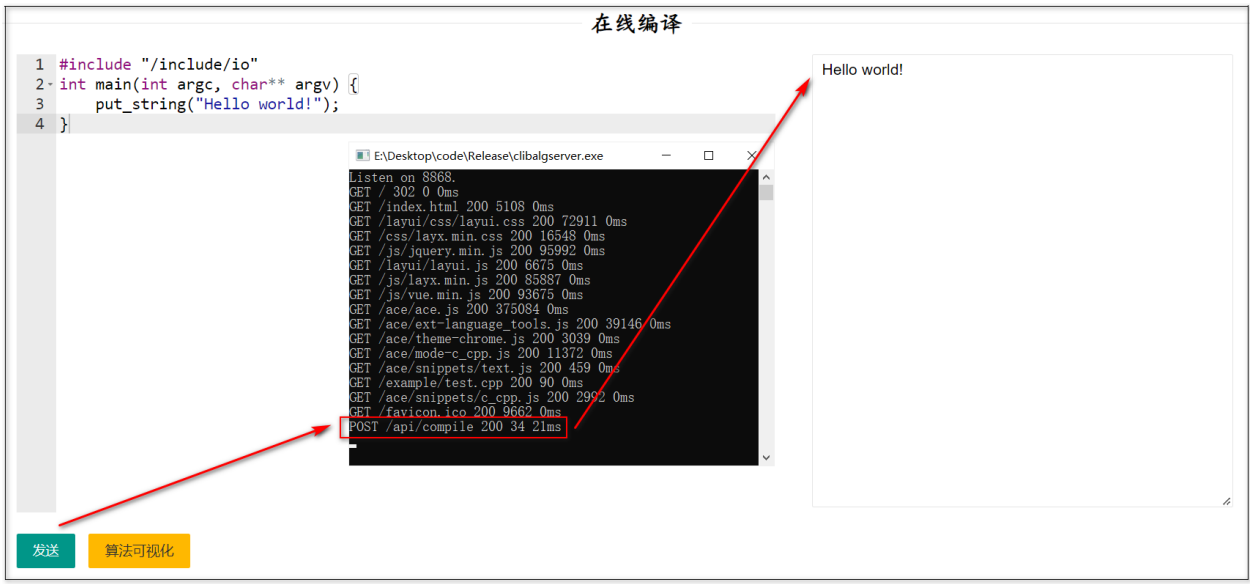

图 24 在线编译的请求处理信息图

5.3 可视化解析模块

5.3.1 流程设计

可视化解析的流程图如图 25 所示。

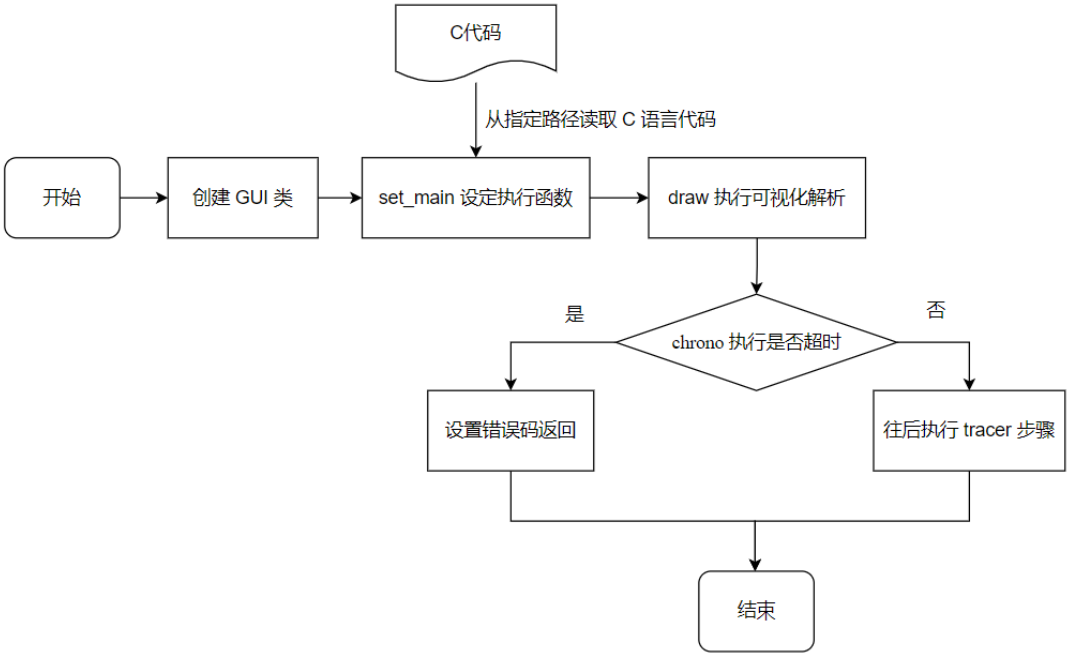

图 25 可视化解析模块流程图

可视化解析的主要实现逻辑，是通过设计的一个 GUI 类进行解析，根据对应的请求，加载服务器端的代码，然后通过 C 编译器来执行 tracer，并将 tracer 步骤的执行结果转化为 JSON，返回给前端。

具体的实现过程如下，在接收到 visulaize 请求之后，handle_API 函数会将请求的代码路径从 JSON 中解析出来，传给 visulaize 函数，该函数内部读取代码路径的代码，使用 Cprase 中的 GUI 类对代码进行可视化。

使用 GUI 类进行可视化时，GUI 类会先把代码读取到内存中，并使用 set_main 函数将代码设置为主代码，再使用 draw 函数对代码进行执行和可视化解析,执行的期间要调用 chrono 时间库中的函数记录执行时间，如果代码死循环了，不能让代码一直执行，这样的处理过程对服务器资源来说是非常浪费的，而且还可能造成系统宕机，所以要设定一个代码执行时间的区间，如果超时就设定一个错误码，继续往后执行 tracer 部分。

处理 tracer 的流程图如图 26 所示。

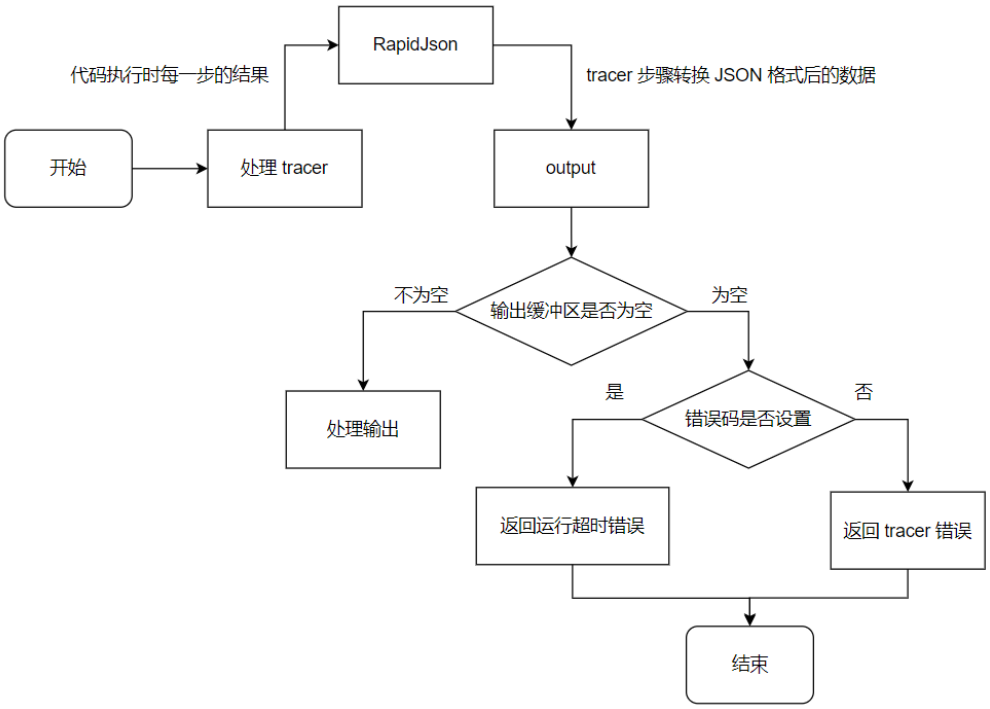

图 26 处理 tracer 流程图

对于 tracer 部分，使用 GUI 类中的 tracer_json 函数对代码中的每一步进行跟踪并转化为 JSON 格式的数据，然后使用 output 函数，将 JSON 格式的数据读取到输出缓冲区，最后函数返回的时候对错误码和输出缓冲区进行判断，如果输出缓冲区为空则返回 tracer 错误，如果错误码被设置，则返回运行超时错误，因为输出缓冲区为输出型参数，在函数的形参中已经被设置，并且在 handle_API 处理函数中已经传入，所以返回值，只需要返回该函数执行的状态码，是否是成功执行，即可，后续的处理则会继续交给 handle_API 来处理

该部分最重要的设计是 GUI 类中 tracer_json、draw、Tick 函数的设计。其中 draw 函数，是通过调用 Tick 函数来执行对应代码并记录 tracer 步骤，每调用一次 Tick 函数就记录一次 tracer，执行代码，并将执行的结果转化为图形格式。

Tick 函数的主要功能为，判断执行后的可执行文件是否存在，如果存在那么代码已经执行完成，则不需要继续执行，如果不存在，则判断该代码是否在执行中如果在执行中则对其进行异常判断，如果异常则抛出异常，否则正常跟踪执行，执行完成之后将 running 状态码设置为 false，表示当前代码执行完成，如果代码没有在执行中，则调用 compile 函数执行代码，然后将代码文件加载进入内存，等待代码执行完成将 running 状态设置为 true 表示代码开始执行。

Tick 函数记录 tracer 的流程图如图 27 所示。

图 27 Tick 函数记录 tracer 流程图

执行完 draw 函数之后，会生成一个 trace_recods 表，存储的是执行完代码图形化之后的数据，此时再调用 tracer_json 函数将该表转化为 JSON 格式的数据返回给前端即可完成代码的可视化。tracer_json 的主要工作就是调用 RapidJSON 库中的 JSON 转换函数，将 draw 函数生成的图形化数据，再将其转化为 JSON 数据，具体的实现逻辑如下：判断 trace_recods 是否有数据，如果有数据，就循环读取表中的数据，并对数据的进行解析分类，最后插入到一个 JSON 格式的文档对象中，将这个对象解析，以字符串对象形式返回即可。

draw 函数转化 JSON 的流程图如图 28 所示。

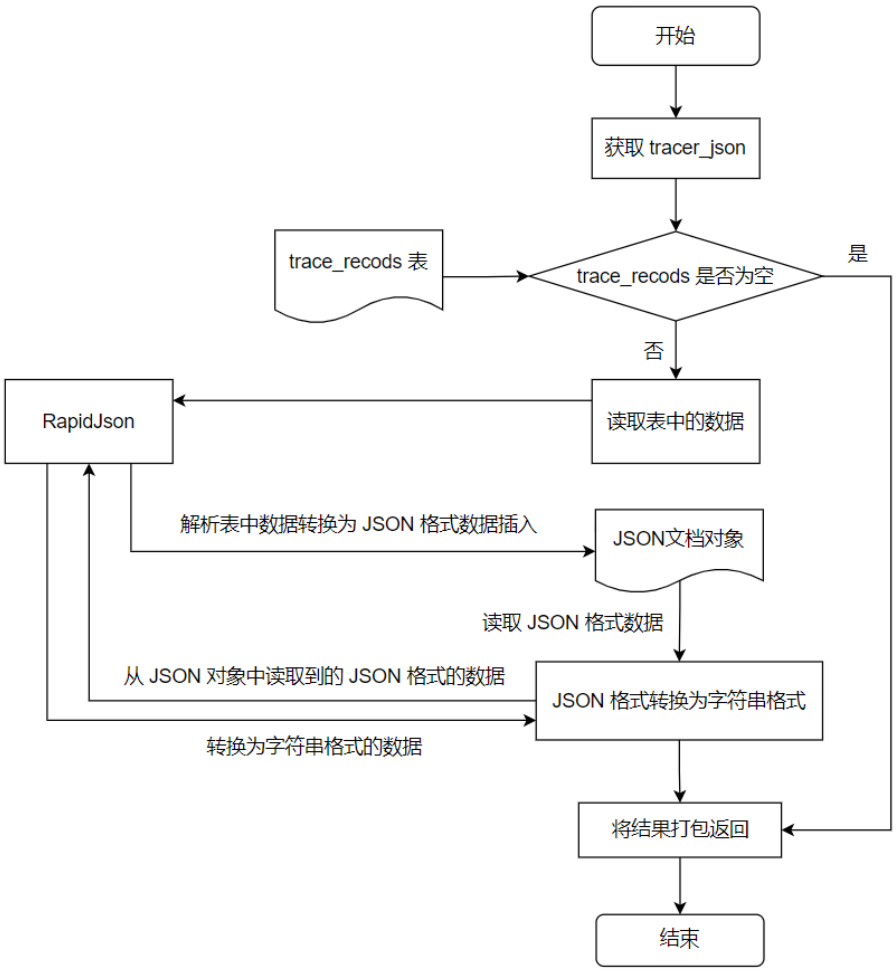

图 28 draw 函数转化 JSON 流程图

5.3.2 核心代码

可视化解析中最核心的函数为 draw 函数，其核心代码如下：

```
void cGUI::draw(bool paused, decimal fps) {
    if (!paused) {
        if (cvm::global_state.interrupt)                //获取全局中断
            cycle = GUI_CYCLES;                         //循环可视化解析
        else if (cycle_stable > 0) {
            if (fps > GUI_MAX_FPS_RATE)                 //判断可视化的帧数
                cycle = __min(cycle << 1, GUI_MAX_CYCLE);       //找到最小循环次数
            else if (fps < GUI_MIN_FPS_RATE)
                cycle_stable--;}                        //若可视化帧数小于最小帧数，则更新最小帧数
    } else if (fps > GUI_MAX_FPS_RATE) {
```

```
        if (cycle_speed >= 0) {
            //设置循环的速度为可视化最小速度和循环速度中较小的一个
            cycle_speed = __min(cycle_speed + 1, GUI_MAX_SPEED);
            cycle = __min(cycle << cycle_speed, GUI_MAX_CYCLE);
        }
        else cycle_speed = 0;           //如果循环速度为 0 就说明循环结束
    } reset_ips();
    for (int i=0; i<ticks+cycle_speed; ++i)
        Tick();                         //重新调用 Tick 进行检测执行代码
}
```

5.3.3 实现效果

可视化解析模块的核心功能就是转化数据，最终结果是将 JSON 格式的数据返回，算法代码在经历了前面的处理之后，所产生的数据会交给动态渲染模块处理，动态渲染模块会将这些数据全部转化为对应的 JSON 格式的数据，再交由 Chart.js 一条一条渲染。

如图 29 所示，对“测试变量”算法进行渲染，右边的输出结果中只显示了 4 条 JSON 格式的数据，“测试变量”算法对应的 JSON 数据如图 29 所示。

图 29 “测试变量”对应的可视化解析结果图

算法的复杂性直接关系到渲染数据的复杂性，Dijkstra 算法明显会比测试案例更加复杂，所以要渲染的 JSON 数据也更多， Dijkstra 算法对应的 JSON 数据如图 30 所示。

图 30 Dijkstra 算法的可视化解析结果图

5.4 动态渲染模块

5.4.1 流程设计

前端申请了一个全局类型的变量注册在浏览器环境中，即 global 变量，会一直等待后端发送的 JSON 数据，在获取到后端发送的 JSON 数据之后，Chart.js 就可以将其渲染成具体的图表，虽然 Chart.js 本身是静态渲染的，但 global 变量会监视变化，后端每发送一次 JSON 数据，Chart.js 就会重新渲染一次，Chart.js 本身是及其轻量的，所以频繁的请求也不会造成渲染过程的卡顿，动态渲染过程仍然是流畅的。动态渲染的流程图如图 31 所示。

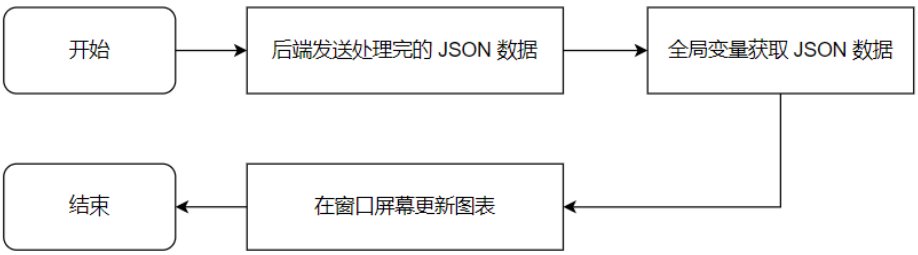

图 31 动态渲染流程图

5.4.2 核心代码

```
(function (global, factory) {                    // 立即执行指向全局对象的函数
    typeof exports === 'object' &&
    typeof module !== 'undefined' ? module.exports = factory(require('chart.js')) :
    typeof define === 'function' && define.amd ? define(['chart.js'], factory) :
    (global = global || self, global.ChartDataLabels = factory(global.Chart));
} (this, function (Chart) { 'use strict';          // 引用 Chart 对象，使用严格模式
    Chart = Chart && Chart.hasOwnProperty('default') ? Chart['default'] : Chart;
    var helpers = Chart.helpers;                   // 提供用于生成图表的扩展方法
    var devicePixelRatio = (function() {
      if (typeof window !== 'undefined') {  // 如果检测执行的环境不存在 window 对象
        if (window.devicePixelRatio)
          return window.devicePixelRatio;
        var screen = window.screen;          // 窗口对象，包含屏幕信息
        if (screen)
          return (screen.deviceXDPI || 1) / (screen.logicalXDPI || 1);
      }
      return 1;
    }());
}
```

5.4.3 实现效果

经过了之前的三个模块，一个算法最终会被转化为若干条 JSON 格式的数据，而动态渲染模块就是要实现最后一步，也就是将这些 JSON 数据经过 Chart.js 渲染成可视化的图表，动态渲染的过程实际上就是不断地更新渲染 JSON 数据格式的过程。

我们以最开始的“测试一维数组”算法为例，点击“发送”，我们就可以查看渲染的过程，动态渲染时，首先会在原窗口的基础上生成新的窗口，这个窗口包含着一些展示的信息以及几个可操作的按钮。如图 32 所示就是“测试一维数组”算法对应的动态渲染的初始窗口，“当前算法步骤”指的是当前的渲染过程进行到了总过程的第几步，图 32 中显示的是“0/205”，即“测试一维数组”总共有 205 个渲染步骤，我们还没有开始渲染。“当前操作”指的是下一步的动态渲染对应到该算法中的具体步骤，另外，图 32 也表明还没

有开始渲染时所对应到的算法步骤就是要输出一维数组修改示例。“测试一维数组”算法对应的动态渲染的初始窗口如图 32 所示。

图 32 对“测试一维数组”动态渲染的初始窗口图

下面有四个按钮，“继续”代表着只往下渲染一步，“自动”指的是自动渲染所有步骤，“重置”，“结束”指的是立刻结束渲染并关闭窗口。

如图 33 所示，就是我们对“测试一维数组”算法点击“继续”按钮的效果图，可以看到，“当前步骤”由“0/205”变为了“1/205”，当前操作也由“输出：一维数组修改示例”变为了“创建整型数组：数组[10]”，此外，也多了一个小窗口，这个小窗口就是动态渲染所产生的新的图表。对“测试一维数组”算法初始窗口执行“继续”如图 33 所示。

图 33 对“测试一维数组”初始窗口执行“继续”效果图

我们点击“自动”，那么我们将会看到“测试一维数组”对应的 205 步动态渲染的完整展示，我们暂停到第 127 步时的展示结果如图 34 所示。

图 34 对“测试一维数组”暂停到第 127 步时的渲染效果图

“测试一维数组”最终的渲染结果如图 35 所示。

图 35“测试一维数组”最终渲染效果图

此外，所有生成的窗口都是可移动的，我们仍然以最开始的“测试一维数组”算法为例，点击“发送”，即可查看渲染的过程，动态渲染时，会在原窗口的基础上生成新的窗口，这些窗口都是可以移动的，这个功能十分方便使用，我们将图 35 中的窗口全部移动，

移动了窗口之后的渲染效果图如图 36 所示。

图 36 移动“测试一维数组”渲染窗口效果图

动态渲染广度优先搜索算法的最终效果如图 37 所示。

图 37 广度优先搜索算法的最终渲染效果图

对 Kruskai 最小生成树算法动态渲染到第 440 步的效果如图 38 所示。

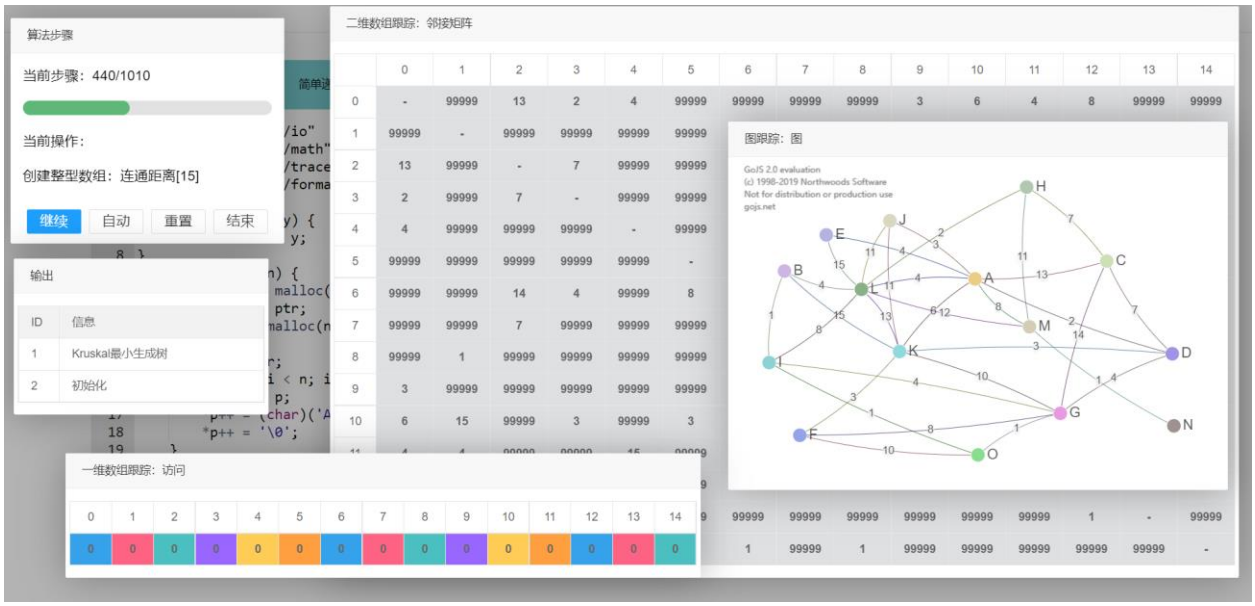

图 38 Kruskai 算法渲染到第 440 步的效果图

动态渲染 Kruskai 算法的最终效果如图 39 所示。

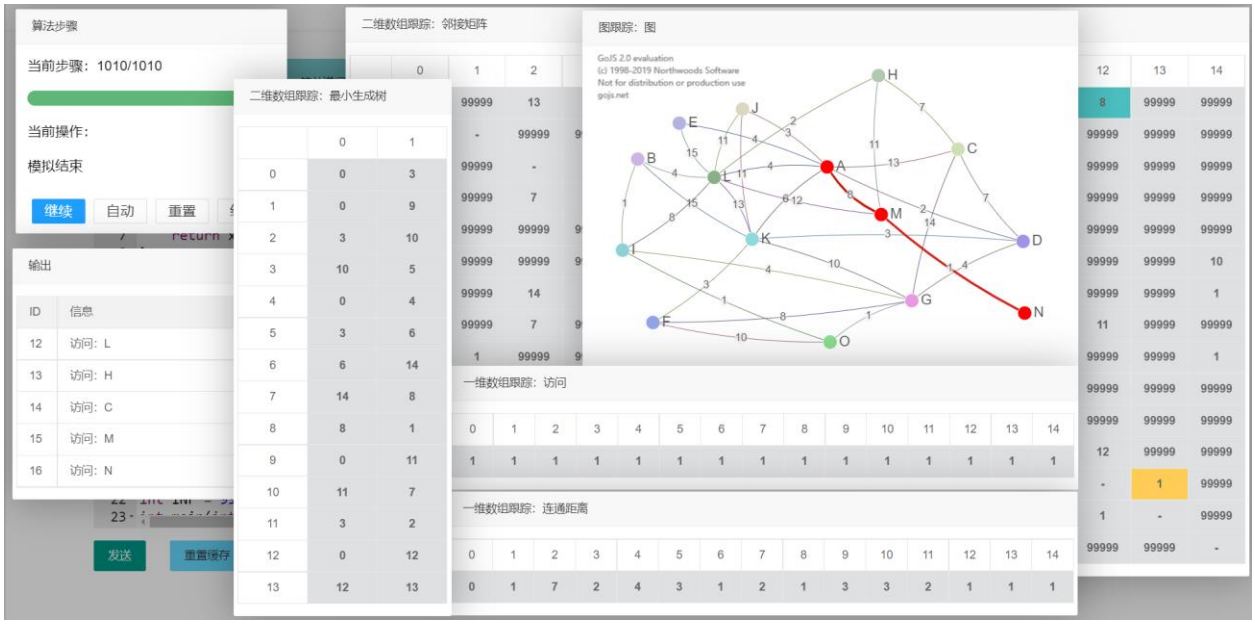

图 39 Kruskai 算法的最终渲染效果图

动态渲染 Floyd 最短路径算法的最终效果如图 40 所示。

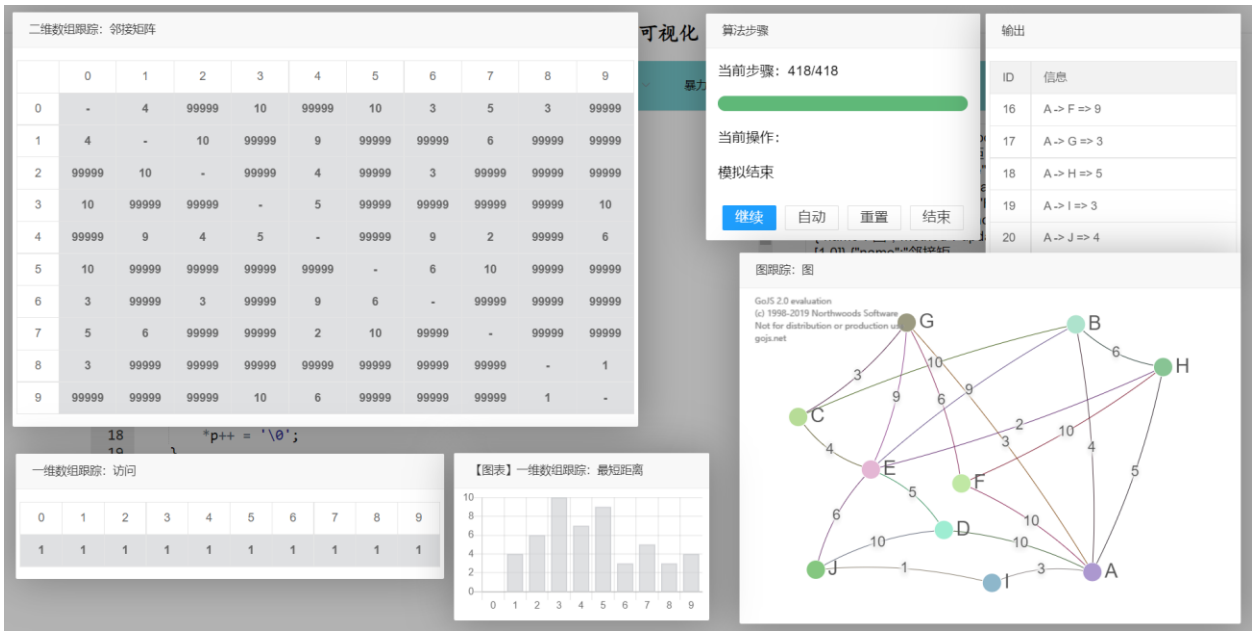

图 40 Floyd 最短路径算法的最终渲染效果图

5.5 系统算法库及其动态扩展

5.5.1 系统算法库

算法库就是用 C++语法实现的各种算法的源代码库，系统现如今的基础算法库一共包含了 8 个算法主题、42 种算法，用户可以在导航栏中选择算法，“代码框”会显示对应的源代码。在“排序算法”主题中选择“冒泡排序”算法如图 41 所示。

图 41 选择“冒泡排序”算法图

在“图相关算法”主题中选择“深度优先搜索”算法如图 42 所示。

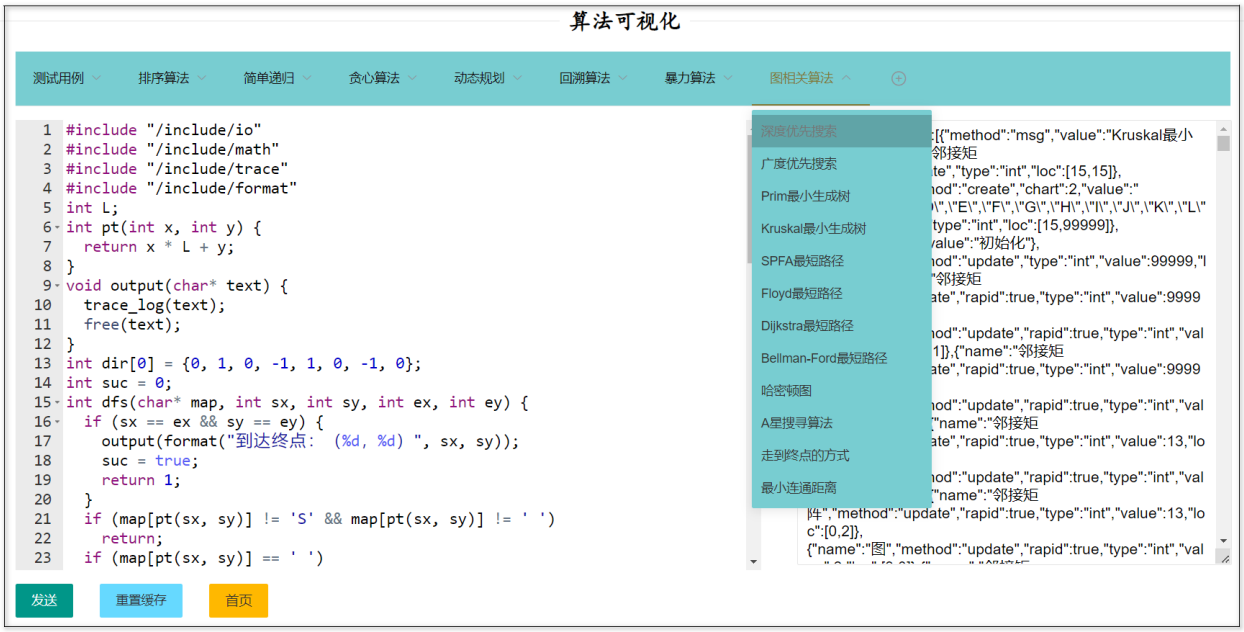

图 42 选择“深度优先搜索”算法图

算法库中的算法可以直接渲染，方便用户直接拿来学习，例如用户选择“Prim 最小生成树”算法，最后的渲染结果如图 43 所示。

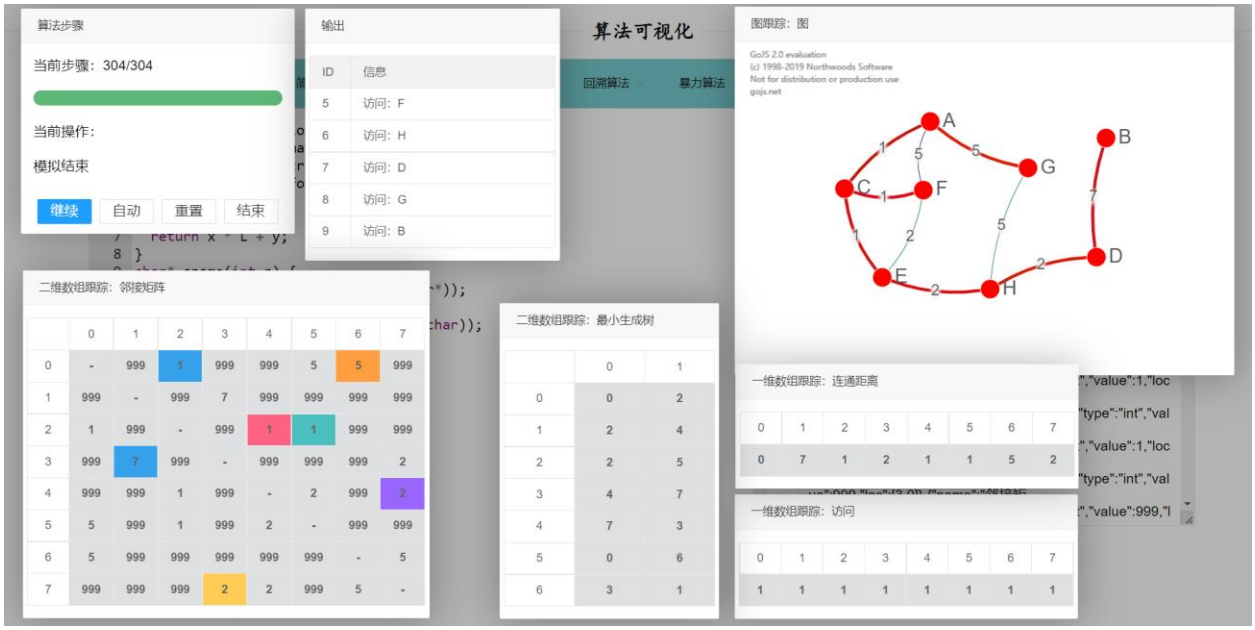

图 43“Prim 最小生成树”算法渲染结果图

算法库的设计目的是更好地方便用户对一些算法“开箱即用”，除此之外，因为系统本身拥有在线编译的功能，所以对于那些用户自己写的代码而非算法库中的代码，系统也可以在线编译之后动态渲染出来。我们以算法库中原有的“简单递归”主题中的“第 N 个阶乘”算法为例。算法库的“第 N 个阶乘”算法中，分别定义了”n=20”以及”a[20]”，它的动态渲染结果一共有 45 步，如图 44 所示。

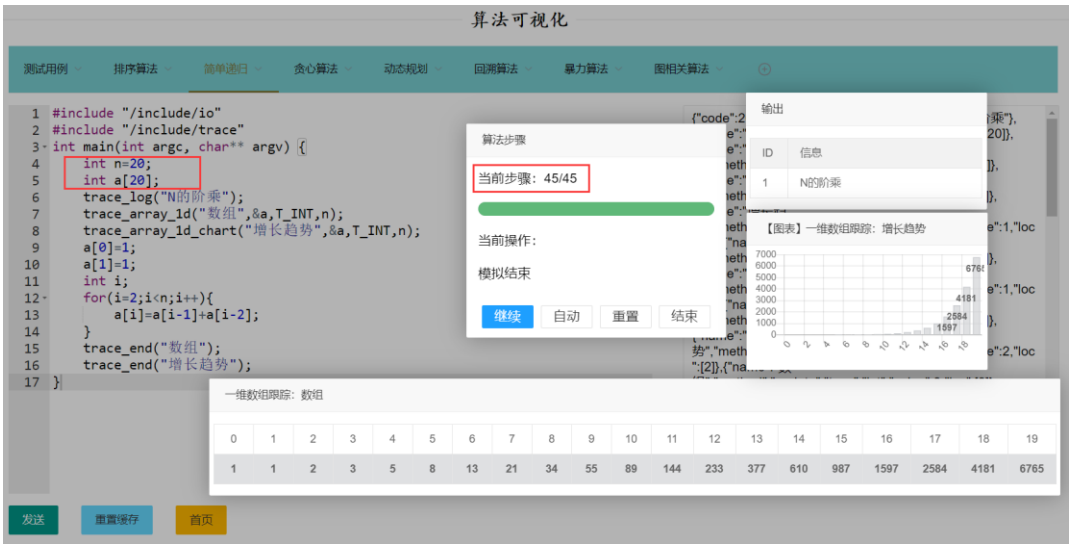

图 44 算法库中的“第 N 个阶乘”算法渲染结果图

之后我们可以对代码进行修改，我们将”n=20”和”a[20]”分别改为”n=10”和”a[10]”，则修改之后的渲染结果一共有 25 步，而且对应的渲染的结果也会发生变化，如图 45 所示。

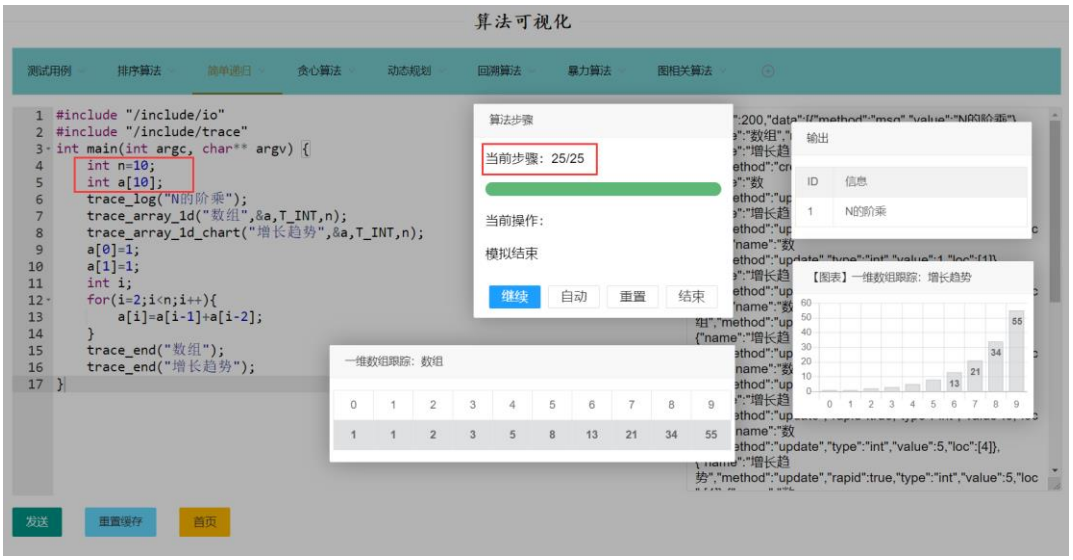

图 45 对“第 N 个阶乘”算法修改之后渲染结果图

5.5.2 算法库的动态扩展

为了方便用户的学习和使用，系统除了拥有原有的基础算法库之外，也允许用户上传自己的文件到系统中，即系统的算法库具有动态扩展的特性。

具体的操作步骤是，用户点击顶部导航栏中最右边的图标，便可以在弹出框中新增一个自己命名的算法主题，选择算法主题之后，可以再在该主题下新增一个自己命名的算法，上传文件之后，系统会将该算法名称与文件相互绑定，此时用户就在系统的基础算法库中新增了属于自己的算法。

点击顶部导航栏中最右边的图标，新增按钮的图标样式如图 46 所示。

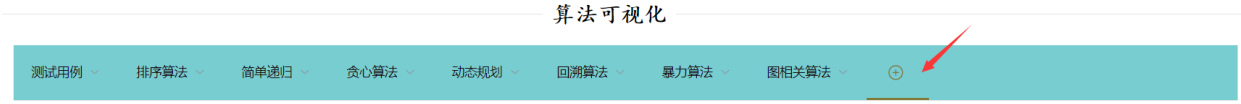

图 46 新增算法的按钮图标样式图

点击此按钮，会出现一个弹出框，用户可以新增自己的算法主题，也就是在导航栏中新增一个一级菜单，点击新增按钮图标之后的弹出框如图 47 所示。

图 47 新增算法主题弹出框样式图

点击“添加算法主题”之后会在导航栏中动态生成一个一级菜单栏，如图 48 所示。

图 48 新增算法主题之后的导航栏样式图

但此时我们只是新增了一个算法主题，还没有在此主题下新增任何算法，我们将鼠标移动到“新的算法主题”栏目，下拉框中自带有“添加算法”的按钮，如图 49 所示。

图 49 “新增算法主题”菜单栏的下拉框按钮图

我们点击“添加算法”按钮，又会弹出一个新的窗口，如图 50 所示。

图 50 新增算法弹出框样式图

此时我们可以在窗口中输入我们要新增的算法名称以及选择对对应的算法文件，默认输入的名称是“新的算法”，这里改为“我的算法”，并且选择 main.cpp 文件，如图 51 所示。

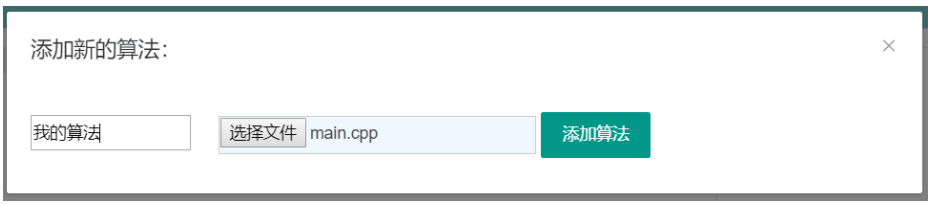

图 51 新增我的算法并选择 main.cpp 文件

点击“添加算法”之后，就会在该菜单栏中显示出对应的算法，如图 52 所示。

图 52 成功新增“我的算法”样式图

点击“我的算法”，就会在代码框中显示出算法文件的源代码，如图 53 所示。

图 53 “我的算法”代码显示图

我们仍然可以渲染此代码，对“我的算法”的渲染结果如图 54 所示。

图 54 “我的算法”渲染结果图

为了方便用户使用，我们可以对新增的算法主题的名称进行修改，或者删除新增的算法主题。具体的操作步骤是，双击我们新的算法主题菜单栏，会弹出一个窗口，在窗口中进行修改或删除。

由于在之前的步骤中，我们已经新增了算法主题，现在我们就双击“新增的算法主题”，弹出的对应窗口如图 55 所示。

图 55 修改或删除算法主题的弹出框样式图

我们在输入框中输入“新增的算法主题 222”，再点击“修改算法主题名称”，则导航栏会立即更新对应算法主题的名称，如图 56 所示。

图 56 将算法主题名称修改为“新增的算法主题 222”结果图

若我们直接点击“删除算法主题”，则导航栏会同步删除该菜单栏，如图 57 所示。

图 57 删除“新增的算法主题 222”之后的导航栏样式图

6 总结与展望

6.1 研究总结

市面上针对算法可视化的平台本身就比较少，带有交互式设计的平台就更少了，但算法的可视化教学在辅助教学中的意义是十分重要的，并且算法与数据结构是电子信息类学科尤其是计算机专业的必修课，是重中之重，开发一套可交互的算法可视平台非常有利于提升算法教学课程质量的，这也有利于进一步提高师生对这门十分抽象的课程的兴趣，所以能将算法可视化呈现并且带有交互式功能的平台是非常有必要开发的。

文章从算法与数据结构的可视化教学的各种实际需求出发设计平台，以算法可视化交互式平台为研究对象，采用了 HTML、JavaScript、Chart.js、Libevent、RapidJSON 等技术，对算法可视化交互式平台的请求处理模块、代码编译模块、可视化解析模块、动态渲染模块以及其他小功能模块等都进行了详细的设计。

在便捷性使用方面，系统借助 C++开发，最终可打包成一个可执行文件，相较于其他系统依赖于环境、网络以及登录功能，系统并不完全依赖于浏览器，而是从可执行文件启动，从而摆脱了对环境配置的依赖，在浏览器使用，也不需要网络，也不需要管理账号和密码，真正做到了开箱即用。

在编辑功能方面，系统从设计结构角度优化实现方式，最终做到了媲美市面上主流编辑器的轻量编辑器功能，用户可借助各种编辑功能完成高效学习，很大程度上提高了学习效率。

在编译功能方面，系统能绕开了主流编译器那些庞大复杂的编译架构，而是采用小巧轻便的设计方式，只针对代码编译，这样也大幅度提高了在线编译的效率，同时考虑到编译缓存，系统也直接提供了重置缓存的功能方便用户使用。

在动态可视化渲染方面，系统同时兼顾视觉效果设计以及可控渲染设计，不仅用各种图表以及高亮颜色来展示算法的动态渲染过程，还为用户提供了控制窗口，用户也随时开始、暂停、继续、重置渲染，也可以直接选择自动渲染，这极大地增强了算法学习趣味性，将抽象的算法过程动态可视化渲染出来也让用户对算法心生畏惧。同时，借助系统本身的在线编译功能，用户可以自己在代码框中更改代码，再实时渲染自己的代码，这也十分便于用户将所有精力投入到算法学习本身。

在系统的可扩展性方面，系统除了具备庞大的基础算法库之外，还支持用户自己新增、删除、修改自己的算法主题，用户只需要上传自己的算法文件到系统中即可。

综上所述，先是从系统功能的实用性角度考量研究方向与主要内容，再从设计实现的角度出发优化设计架构，最后结合相应的技术知识实现具体功能，相对于国内外的其他算法可视化平台，系统不仅在可视化功能上基本达到了相应效果，也在大部分平台所缺乏的交互式设计方面完善了相关功能，同时，这也促进了国内外在轻量编辑器、在线编译器以及可视化系统教学等方面的进一步研究。

6.2 未来展望

可视化算法交互式平台本身已有的功能已经能较好地辅助算啊教学，在以后的算法教学实践中，不仅可以将系统普及应用到算法教学的课堂上，让学生与老师都体验到算法这

门课的趣味性，也可以为其他学习算法者提供帮助。

由于笔者经验水平的不足，系统仍然有一些没有考虑周全而有待完善的地方，在最开始分析需求时考虑到C++是算法学习的首选语言，再加上时间不充足的原因，目前系统只能渲染C++代码对应算法，在未来的后继开发过程中，将会更加考虑到用户的实际需求完善交互式功能，同时尽可能地实现其他语言的动态可视化渲染。在此希望各位老师可以提出宝贵的意见和建议。

参考文献

[1] 高校课堂教学中大学生自主学习的缺失与重构[J]. 刘颖,沈伯雄. 黑龙江高教研究. 2020(02)

[2] 高校课堂教学质量及评价标准新论[J]. 赵作斌,黄红霞. 中国高等教育. 2019(08)

[3] 直观可视化教学方法在现代高等教育课堂教学中的应用[J]. 杨永明,李霄. 高教学刊. 2022(03) E 2019)(Advances in Economics, Business and Management Research,VOL.110).,2

[4] Qin Liu. The Application of Visual Teaching Resources in Chinese Teaching in Schools for the Deaf[C]//.Proceedings of 5th International Conference on Economics,Management,Law and Education(EML 019:1157-1160.DOI:10.26914/c.cnkihy.2019.044089.

[5] Liu Dan,Shen Nan. Research and Practice of Visual Teaching in Art Design Specialty Based on the Cultivation of Innovative Talents[J]. Frontiers in Educational Research,2021,4.0(15.0).

[6] Aristov Michael M.,Moore John W.,Berry John F.. Library of 3D Visual Teaching Tools for the Chemistry Classroom Accessible via Sketchfab and Viewable in Augmented Reality[J]. JOURNAL OF CHEMICAL EDUCATION,2021,98(9).

[7] 基于文本大数据的学科知识可视化教学探索与实践——以情报学为例[J]. 夏一雪. 现代情报. 2022(02)

[8] 基于大数据的高校智慧课堂可视化教学平台设计[J]. 杨倩倩,王龙. 数字通信世界. 2021(05)

[9] 朱良学.学习可视化表征工具软件算法设计研究[J].湖北师范大学学报(自然科学版),2022,42(01):57-63.

[10] 马晓萍.基于 Unity3D 的冒泡排序算法动态可视化设计及实现[J].喀什大学学报,2021,42(06):66-69.DOI:10.13933/j.cnki.2096-2134.2021.06.012.

[11] 夏齐鸣,刘晓强.AI 算法可视化辅助教学系统的设计和实现[J].电脑知识与技术,2020,16(14):41-43+46.DOI:10.14004/j.cnki.ckt.2020.1483.

[12] 基于“Vue.js”前端框架技术的研究[J]. 方生. 电脑知识与技术. 2021(19)

[13] Vue.js 在前端开发应用中的性能影响研究[J]. 唐斌斌,叶奕. 电子制作. 2020(10)

[14] Web 开发主流框架技术研究[J]. 方阿丽. 无线互联科技. 2021(08)

[15] 曹灿,刘志刚. 基于 SSH 和 Layui 的工程科学前沿与实践系统[J]. 工业控制计算

机,2019,32(2):91-92,96. DOI:10.3969/j.issn.1001-182X.2019.02.040.

[16] 尹胜燕.基于 LayUI 框架的学生社团管理[J].大众投资指南,2019(05):129.

[17] 曹灿,刘志刚.基于 SSH 和 Layui 的工程科学前沿与实践系统[J].工业控制计算机,2019,32(02):91-92+96.

[18] 梁钧儒. 浅析 AmazeUI 框架下结合 datatables 表格和 layui 组件开发 web 后台界面制作过程[J]. 商情,2019(38):168.

[19] 杨勇. 基于 LayUI 框架的前端网页跨域问题的研究[J]. 数字化用户,2019,25(51):128.

[20] 改进 LR 算法的汉语层次化句法分析器设计[J]. 皮乾东,邵玉斌,龙华,杨陈菊. 控制工程. 2021(12)

[21] 《编译原理》中 LR（0）语法分析动态演示系统分析与设计[J]. 石凤贵. 电脑知识与技术. 2020(03)

[22] 任小强,王雪梅,唐晓华,等.基于 Python 的编译原理教学演示模块设计与实践[J].工业控制计算机.2021,(9).doi:10.3969/j.issn.1001-182X.2021.09.028.

[23] 栈在编译程序语法分析中的应用[J]. 朱朝霞. 电脑知识与技术. 2017(17)

[24] 一个诠释编译理论的解释器 Tina[J]. 房超,杨连贺. 电脑知识与技术. 2018(14)

[25] 蒋瀚如. 并发程序分离编译的验证[D]. 安徽:中国科学技术大学,2019.

[26] 贾伟涛,回宝瑞. 基于Libevent的网络通信框架设计与实现[C]//.第三十四届中国（天津）2020’IT、网络、信息技术、电子、仪器仪表创新学术会议论文集.,2020:332-336.DOI:10.26914/c.cnkihy.2020.010113.

[27] 代军普. 基于 Libevent 的栅格数据服务引擎设计与实现[J]. 山东理工大学学报（自然科学版）,2016,30(4):70-74. DOI:10.3969/j.issn.1672-6197.2016.04.017.

[28] 程瑞. 基于 Libevent 架构的终端安全接入管理系统的研究与实现[D]. 华北电力大学,2017. DOI:10.7666/d.Y3264435.

[29] 国网甘肃省电力公司电力科学研究院,国网甘肃省电力公司,国家电网公司,等. 一种基于 Libevent 架构的新能源厂站发电单元终端接入管理系统:CN201810924809.1[P]. 2019-01-25.

[30] 贾伟涛,回宝瑞. 基于Libevent的网络通信框架设计与实现[C]//.第三十四届中国（天津）2020’IT、网络、信息技术、电子、仪器仪表创新学术会议论文集.,2020:332-336.DOI:10.26914/c.cnkihy.2020.010113.

[31] 达虎,柳亭,封蕾,李葆光,田全红.基于 B/S 架构的开放基金管理系统设计[J].微型电脑应

用,2022,38(02):16-18.

[32] 蒋豪博,刘志平,张恒,许志刚.基于Qt与D3.js的新型交互式图表开发[J].淮海工学院学报(自然科学版),2017,26(02):21-25.

[33] 刘班.基于 JS Charts 实现 Web 应用数据的可视化[J].电子技术与软件工程,2019(20):152-153.