国产编程语言蓝皮书
2024

编程语言开放社区（PLOC）
湖北省软件行业协会

版权声明

- 特别顾问：梁宇宁、李建忠
- 策划：柴树杉、丁尔男、李智勇、杨海龙
- 编辑：陈朝臣、李登淳、李皓琨、吴森、徐芳、杨海玲、赵普明、朱子润
- 支持单位：GitCode
- 赞助单位：武汉凹语言科技有限公司

对编程爱好者之间，尤其是应用程序开发者和编程语言开发者之间的交流，蓝皮书发挥着重要的作用。随着时间的推移，搜集各种编程语言并根据它们发展过程中优缺点的变化进行讨论是一个很好的主意。在计算机时代刚开始时，计算机科学家寥寥无几，他们通常对从底层硬件到实际应用的各个环节都了解甚多，那时的编程语言是为专家设计的；自互联网时代拉开序幕以来，编程语言从复杂演变得简单，比如深受新手程序员欢迎的 JavaScript，它被用于开发网页应用程序而非复杂的科学应用程序。现在人工智能的时代开始了，我们可以预见，新的编程语言可能将面向人工智能模型设计，而不像过去的语言那样面向人类程序员设计！在未来的人工智能发展中，《国产编程语言蓝皮书》的价值将愈发凸显，其重要性不言而喻。希望各位继续努力，祝一切顺利。

——梁宇宁

编程语言是人类和计算机的交汇点，在整个计算产业中扮演着不可或缺的角色。从 1957 年第一门高级编程语言 FORTRAN 诞生后至今近七十年的历史，计算机产业孕育了数百种编程语言，至今被大众使用的主流二十多种。如同数千种的人类语言演化史，有着异曲同工之妙。

编程语言也是技术变革的催化剂。每一代技术变革的同时，都有和该时代相应的编程语言孕育而生或蓬勃发展，比如 C++之于系统软件，JavaScript 之于 Web，Java/Go 之于云原生，Python 之于机器学习，等等。

在大语言模型驱动的智能变革时代，由大模型带来的代码自动生成技术正在开创全新的开发范式，为编程语言的发展打开了前所未有的想象空间。

国产编程语言在众多专家、学者、工程师呕心沥血的努力下方兴未艾，是国产软件发展宏图中一道靓丽的风景。《国产编程语言蓝皮书》的编制对于促进国产编程语言的交流和发展非常有意义。让我们携手一起支持国产编程语言的发展！

——李建忠

目 录

第一章　前言

1.1 编制背景

编程语言是软件业的工业母机、编译器技术是信息产业的根技术，各种编程语言被用于操作系统、数据库管理系统、网络服务、工控设备、应用程序等的开发，渗透到了所有现代产业和服务领域。尤其是信息产业创新空间的持续扩展、系统复杂度的持续上升、开发成本的持续降低，都直接受益于不断涌现的编程语言和编译技术。迄今为止，国内几乎没有出现被广泛使用的编程语言，这与我国世界性工业大国、科技大国的地位相去甚远。

工业和信息化部发布的《“十四五”软件和信息技术服务业发展规划》中提到，应“强化基础组件供给……加快突破编程语言开发框架”；中国软件行业协会发布的《中国软件根技术发展白皮书（基础软件册）》第四章专门对编程语言和编译器的重要性、发展态势等进行了归纳。这些文件说明编程语言相关产业的发展获得了政策支持。信息技术在我国经过多年发展积累，已形成从业人数近千万的大型产业，对编程语言这一基本工具的需求本就非常强烈；而大语言模型、国产芯片等新兴方向的井喷式增长更是对编程语言提出了很多全新的需求。

回顾历史不难发现，与其他产业不同，作为信息产业的核心，编程语言的成功案例充满了偶然性。目前广泛使用的编程语言和开发工具，既有由大型企业推动的商业项目，也有由个人发起的开源项目；既有以 KPI 为驱动的商业产品，也有由兴趣驱动的产品。当前国内的根软件行业也正呈现出项目高度分散的趋势，企业、开源社区发起了大量不同类型、用于不同领域的新兴编程语言项目。

1.2 编制目的

基于上述背景，PLOC 发起编撰并发布《国产编程语言蓝皮书》（即本文，以下简称蓝皮书），力争全面的收纳国内已具备一定可用性的、活跃的编程语言项目，为业界提供一份客观的国产语言全景图。我们希望蓝皮书尽可能客观的反应国内实用型编程语言项目的总体情况，为行业内外提供全局视角，协助工业界需求方寻找合适的语言、帮助编程语言爱好者寻找可参与贡献的开源项目。蓝皮书将定期发布，以追踪行业最新进展。

“从业者互助”是 PLOC 社区的精神内核，蓝皮书延续了这一特点。本文中收录的项目均为自主申报，编委对项目资格进行审核；项目展示内容（文字、图片等）由项目方提供，编辑仅对页面版式进行调整。最了解语言特性的人是语言作者，我们希望通过自主申报，让各项目的特点以最符合作者个性的形式得到展现，以期吸引到趣味相投的爱好者、贡献者、潜在使用者。

为保持信息时效性，蓝皮书将持续更新发布。《国产编程语言蓝皮书-2024》是蓝皮书第二版，根据 2023 版编撰经验和各方反馈，2024 版有以下更新：

- 增加英文版。最终将分别发布中英文双语版本，分别成册，内容一致；
- 申报资料中的“项目简介”部分增加了建议内容清单，利于项目展示标准化。

1.3 收录标准

符合以下条件的项目可在蓝皮书工作区仓库中通过 PR 发起申报：

1. 项目由国内的企业、社区或个人发起和维护；
2. 项目符合项目分类标准（见 1.5 节）；
3. 项目基本可用，且能被编委会独立验证；
4. 面向公众开放；
5. 项目处于活动状态。

> 《国产编程语言蓝皮书》-2024 编委会对收录标准拥有最终解释权。

1.4 收录方法

蓝皮书中收录的项目均为自主申报，满足收录标准的项目可在以下地址申报：

https://gitcode.com/ploc-org/CNPL/tree/master/projects

在上述目录中增加项目同名目录，将项目简介等资料以 markdown 格式填入其中，资料中应包括以下内容：

- 项目名称
- 项目图标
- 项目主页
- 项目仓库
- 项目分类标签
- 中英文项目简介
- 申报人联系方式

其中项目简介部分，建议包含以下内容，形式不限：

- 目标应用场景
- 项目特点、设计理念
- 简单示例（比如 hello world）
- 项目目标或寄语

未提供英文版简介的项目，编委会将自主翻译并入英文版蓝皮书，由于无法保证翻译精准表达项目内涵，请尽可能提供内容一致的双语版本。中英文版均可插入不超过 4 张图片，图片尺寸为 1920×1080 像素（20×11.25 英寸，96ppi），图片格式可选 JPG 或 PNG。申报示例见：

https://gitcode.com/ploc-org/CNPL/tree/master/projects/sample

蓝皮书发布时将使用 5 号字体，竖向 A4 幅面排版，中英文版分别成册，中文版项目资料页

面总篇幅均应不超过 4 页，英文版不应超过 6 页。最终版式参考：

https://gitcode.com/ploc-org/CNPL/tree/master/projects/sample/sample.doc

发起申报 PR 后，编委会将审核项目资料，期间请保持项目地址及网站等可正常访问、申报人联系方式畅通；编委会委员将与您联系，确认项目资料准确无误，若您对于某些选项该如何填写存在疑问，亦可在此时与编委沟通。当您发起申报时，视同您已获得该项目所有者许可，并授权编程语言开放社区（PLOC）在蓝皮书中展示该项目的名称、图标等信息。

1.5 项目分类方法

语言类项目分类标签：

- 付费/免费
- 开源/闭源
- 通用/专用
- 是否接受社区贡献
- 语言类别（详细清单见 附录-语言类别列表）
- 工具类别（详细清单见 附录-工具类别列表）
- 应用领域（详细清单见 附录-应用领域列表）

工具类项目分类标签：

- 付费/免费
- 开源/闭源
- 是否接受社区贡献
- 工具类别（详细清单见 附录-工具类别列表）
- 应用领域（详细清单见 附录-应用领域列表）

第二章　2024 年度小结

与 2023 版相比，《国产编程语言蓝皮书-2024》收录项目由 15 个增加至 20 个，收录变更情况如下：

- 新增项目 6 个，分别为：GödelScript、金鱼 Scheme、MoonBit、Nasal-Interpreter、PikaPython、XLang；
- 减少项目 1 个：豫言（项目方主动撤回）；
- 更名项目 2 个：Z 语言更名为 Auto 语言、K 语言更名为 Koral 语言

自 2024 年 1 月 1 日起，至截稿时，各项目更新情况如表 2-1 所示：

项目名称	更新次数	项目名称	更新次数
Auto 语言	155	Aya	639
Calcit	168	CovScript 智锐	14
DeepLang	23	GödelScript	7
HVML	32	金鱼 Scheme	244
KCL	505	Koral 语言	25
洛书（Losu）编程语言	73	MoonBit	1550
Nasal-Interpreter	137	NASL	2609
PikaPython	206	青语言	27
狮偶	178	凸语言	324
凹语言	506	XLang	1256

表 2-1

> 由于各项目遵循的配置管理规则不同（如是否使用 Squash 等），上述数据仅表示项目是否处于活动状态，不具备横向可比性。

2024 年，国内编程语言社区活动进步巨大，开源社区组织的线下活动纷纷首次开设编程语言分论坛，包括：

- 2024 年 3 月，第 11 届开源操作系统年度技术会议（OS2ATC），编程语言分论坛，由 PLOC 理事魏永明担任出品人：

图 2-1

- 2024 年 4 月，PLOC 在杭州举行首次线下会议：

图 2-2

- 2024 年 10 月，PLOC 亮相 G-Star 嘉年华，荣获“GitCode 年度十大开源社区”奖：

图 2-3

- 多个 PLOC 理事项目被收录至“G-Star Landscape”：

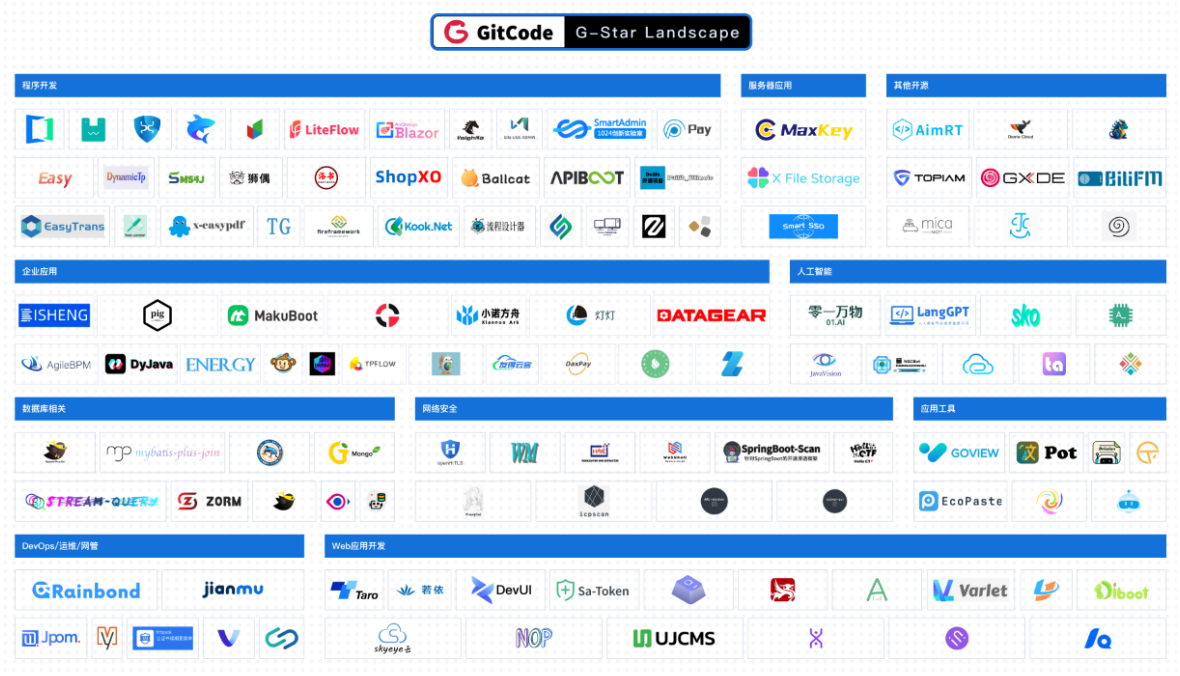


图 2-4

- 2024 年 11 月，第九届中国开源年会暨开源社 10 周年嘉年华，编程语言分论坛，由开源社与 PLOC 联合出品，PLOC 理事会秘书丁尔男担任出品人：

图 2-5

这一系列活动表明，编程语言作为软件工业母机的重要性越发受到各界重视，PLOC 能参与该进程并作出自己的贡献，我们感到由衷的欣慰！

第三章　项目列表

蓝皮书本次共收录项目 20 个，各项目名称如下（按拼音排序，不分中英文）：

- Auto 语言
- Aya
- Calcit
- CovScript 智锐
- DeepLang
- GödelScript
- HVML
- 金鱼 Scheme
- KCL
- Koral 语言
- 洛书（Losu）编程语言
- MoonBit
- Nasal-Interpreter
- NASL
- PikaPython
- 青语言
- 狮偶
- 凸语言
- 凹语言
- XLang

3.1 Auto 语言

项目分类	语言类、免费、开源（MIT）、通用、接受社区贡献
语言类别	通用编程语言
工具类别	解释器、代码生成（C、Rust、Python）
应用领域	行业应用（汽车、嵌入式、机器人、UI）、计算机图形
主页	https://gitee.com/auto-stack/auto-lang
仓库	https://gitee.com/auto-stack/auto-lang

3.1.1 简介

Auto 语言是一门适用于多场景的编程语言。基于 Rust 实现。

Auto 语言的特色有：

- 灵活多变。适应多种生态（C、Rust、Python、Shell 等）。针对不同场景，Auto 语言提供不同的语法、语言特性和标准库；
- 动静皆宜。支持动态类型和静态类型。支持动态解释和静态编译执行；
- 全栈俱备。Auto 语言有自己的标准库、REPL、构建器和 UI 框架。Auto 语言支持前端 UI、后端服务和嵌入式开发。

Auto 语言可以用于下列场景：

- AutoUI：作为 UI 界面的描述语言。

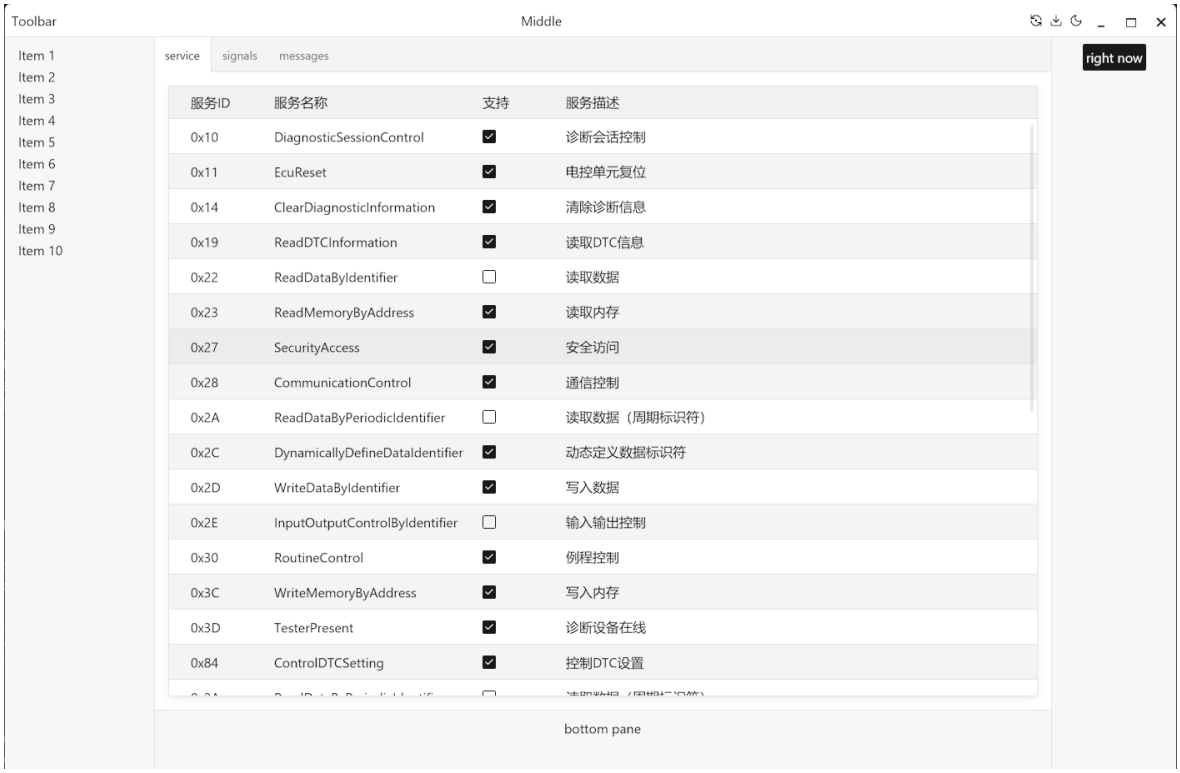


图 3.1-1

```
app {
    left { list(item_list) }
    center {
        tabs {
            tab("service") { service_table(grid) }
            tab("signals") { text("signals") }
            tab("messages") { text("messages") }
        }
    }
    right { text("right pane") }
    bottom { text("bottom pane") }
}
```

- AutoMan：作为配置语言，管理 Auto/C 语言的混合工程。

```
project: "hello"
version: "0.1.0"
// 依赖库自动下载
dep(log, "0.1.0")
// 本工程中的 lib 库
lib("mymath") { link: log }
// 本工程中的 exe 程序
exe("hello") { link: mymath }
```

- 转译为 C 语言，并使用 AutoMan 工具管理 Auto/C 的混合工程。

```
// math.at
pub fn add(a int, b int) int {
    a + b
}
// math.h
#ifndef _MATH_H_
#define _MATH_H_
#include <stdint.h>
int32_t add(int32_t a, int32_t b);
#endif
// math.c
#include <stdint.h>
#include "math.h"
int32_t add(int32_t a, int32_t b) {
    return a + b;
}
```

- 作为 Shell 脚本，实现跨平台的脚本功能。

```
#!auto
cd ~/logs/20241120
grep "error" *.log | wc -l
```

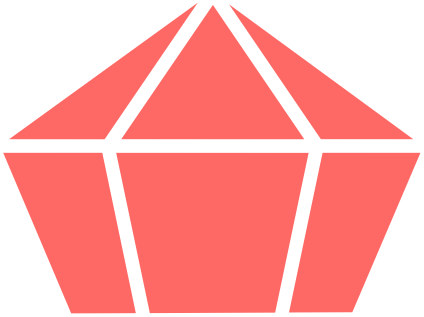
- 作为模版，用于 C/Rust/HTML 等代码的生成。例如：

```
<table>
$ for i, s in students {
    <tr>
        <td>$i: </td>
        <td>${s.name}</td>
        <td>${s.age}</td>
    </tr>
$ }
</table>
```

Auto 语言会自动展开循环，填入 `students` 中的数据，生成完整的 HTML 代码。

- 作为嵌入式脚本，辅助 Rust 的开发。例如，在规划中的引擎项目 `AutoEngine` 中，底层的 3D 图形引擎使用 Rust 的 bevy 引擎，而 Auto 语言则作为脚本语言来管理游戏逻辑。

3.2 Aya



项目分类	语言类、免费、开源（MIT）、通用、接受社区贡献
语言类别	函数式语言
工具类别	解释器、实时编译器
应用领域	通用、计算数学、行业应用(编译器开发)
主页	https://www.aya-prover.org
仓库	https://github.com/aya-prover/aya-dev

3.2.1 简介

Aya 语言是类似 Haskell 和 Lean4 的函数式编程语言，使用归纳类型、模式匹配、一等公民的函数等语言特性作为主要的代码组织工具。

比起 Haskell，Aya 拥有更强大的类型系统，支持『依值类型（dependent type）』，且支持比 Lean4 性质更好的等号类型。换言之，「两个值相等」这件事是一个类型，而它的实例就是这两个值相等的证明。例如，`插入排序 = 归并排序` 是一个合法的类型，并且在 Aya 中它直接等价于函数 `(x : 列表) -> 插入排序(x) = 归并排序(x)`，这个类型的实例需要接收一个列表、返回它被两种排序算法排序后结果相同的证明。这样的证明可以在编程的同时证明一些关于程序的性质。

Aya 团队的招募贴中更详细的项目动机介绍，见：

https://github.com/lazyparser/weloveinterns/blob/master/bunbun

有关 Aya 中语言特性的学术论文参见：

https://www.aya-prover.org/pubs

3.3 Calcit

项目分类	语言类、免费、开源（MIT）、通用、接受社区贡献
语言类别	通用编程语言（脚本语言）
工具类别	代码生成（JavaScript）、 解释器
应用领域	行业应用（网页开发）
主页	https://calcit-lang.org/
仓库	https://github.com/calcit-lang/calcit

3.3.1 简介

Calcit 是 Clojure 的方言，遵循不可变数据结构、前缀表达式、Macros 作为核心设计。使用 Rust 实现，能够快速启动和运行。Calcit 可以直接解释执行，也可以编译为 JavaScript 代码再执行。

生成代码时配合 ES Modules 等现代前端开发习惯进行了简化，相比 ClojureScript 方案更轻量，更易于同 JavaScript 代码混用，也一定程度降低调试成本。

Calcit 的文本形态使用缩进语法。

代码示例一，简单的数据变换，类似 Clojure 语法中的 threading macros：

```
->
  range 100
  map $ fn (x)
    * x x
  foldl 0 &+
  println
```

代码示例二，使用 Macro 封装的基于 Calcit 生态定义的前端 Virtual DOM 组件写法：

```
defcomp comp-inspect (tip data style)
  let
      class-name $ if (string? style) style
      style-map $ if (map? style) style
    pre $ {}
      :class-name $ str-spaced style-data class-name
      :inner-text $ str tip "|: " (grab-info data)
      :style style-map
      :on-click $ fn (e d!)
        if (some? js/window.devtoolsFormatters) (js/console.log data)
          js/console.log $ to-js-data data
```

实际开发中 Calcit 使用数据文件来存储源码。支持使用结构化的方式直接以表达式为单元进行编辑，编辑器内部以数据形态展开，因而也能快速完成部分定义调整和代码重整，从而提升动态类型语言的编写和修改速度：

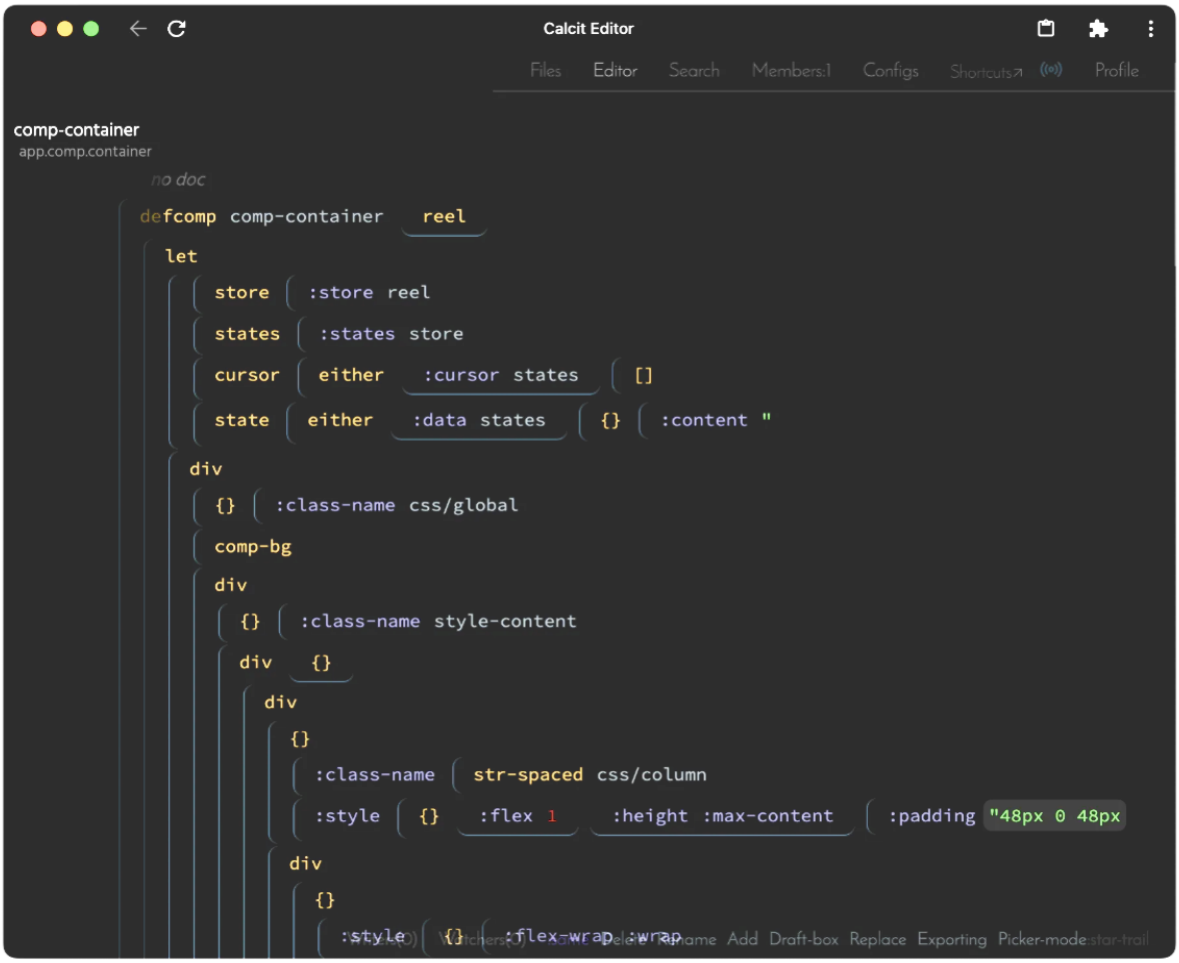


图 3.3-1

Calcit 主要应用于 Web 页面开发场景。实现了部分 Virtual DOM 生态的功能。

3.4 CovScript 智锐

项目分类	语言类、免费、开源（Apache-2.0）、通用、接受社区贡献
语言类别	通用、命令式语言
工具类别	解释器、实时编译器、运行时环境
应用领域	通用
主页	https://covscript.org.cn
仓库	https://github.com/covscript

3.4.1 简介

Covariant Script 编程语言，简称 CovScript，中文名简称智锐编程语言，最初发布于 2017 年，是一门跨平台、开放源代码的动态类型应用层通用编程语言，具有高效、易学、易用、可靠的特点，融合了现代编程语言的优点，可以通过 CNI 高效地与 C++直接交互。

CovScript 编程语言是国内首批投入市场的自主知识产权编程语言之一，具有独立、完善的工具链，包括基础解释器、调试器、实时编译器（JIT Compiler）、标准库、扩展库、文档和 IDE 插件等，不依附于现有编程语言运行时环境。自主、独立、完善且可靠的语言及附属生态使 CovScript 广受客户好评，目前已经在四川大学信息化建设与管理办公室、四川大学华西大数据中心等单位落地，7x24 小时服务于关键系统中。

```
import stdutils

var co = new stdutils.coroutine{[](queue, msg){
    system.out.println(msg)
    foreach i in range(10)
        queue.yield(i)
        if queue.avail()
            system.out.println(queue.get())
        end
    end
    system.out.println("Bye~")
}}

co.join("Hello")
var val = 0
loop
    if co.queue.avail()
        val = co.get()
        system.out.println(val)
    end
until co.resume(val + 1) == stdutils.coroutine_status.finish
```

CovScript 编程语言是以命令式为主体，面向对象和函数式为辅的多范式编程语言，对于初学者来说简单易懂、符合直觉，解决大型项目的需求也能游刃有余。目前 CovScript 已有数个成熟的开发框架：

- CovAnalysis：性能比肩 Pandas 的数据分析、处理框架；
- CSDBC：基于 ODBC 的数据库连接件，兼容绝大多数主流 RDBMS；
- ParserGen：基于类 EBNF 规则的实时语法分析器生成器，CovScript 基于此实现了完全自举。

除此之外，CovScript 还有完善的包管理器和无数协助开发的工具库，能够帮助用户高效的满足大多数云原生应用的需求。

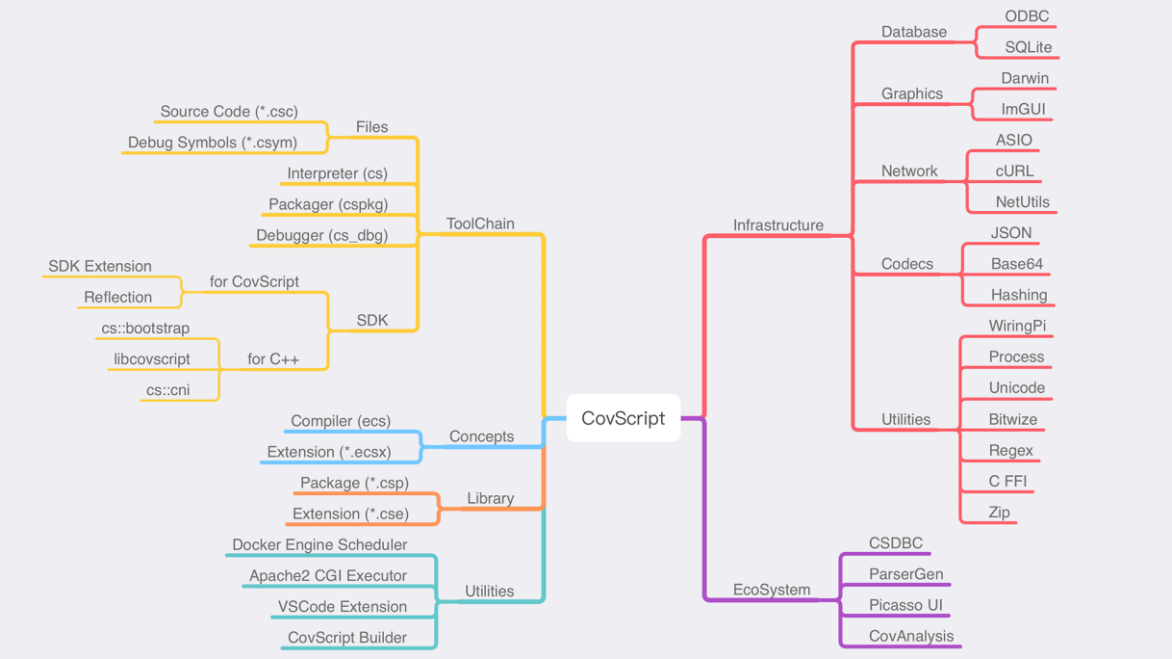


图 3.4-1

CovScript 虽然是一门动态编程语言，但其核心 Runtime 是由高度优化的 C++代码编写而成，其执行速度高达 900 万行代码每秒，协程的上下文切换速度更是达到 80 GOPS（每秒进行的十亿次操作数），能够高效地支持各类应用需求。

作为一门由中国人主导设计、开发的编程语言，CovScript 更是以自身行动践行“中国智造”，其语言核心生态拥有 100%自主知识产权（已在中华人民共和国国家版权局注册，登记号：2020SR0408026；已被 Zenodo 检索，DOI 号：10.5281/zenodo.10471188），周边生态 100%开源、可信。不仅如此，CovScript 还完全支持国产生态：

- 针对龙芯架构和国产操作系统专门优化、测试；
- CovScript 每个发行版都会有对应的龙芯版（UOS@3A4000）；
- 兼容华为鲲鹏处理器和 openEuler 操作系统；
- CSDBC 兼容华为 openGauss 数据库。

为了给尽可能多的客户提供服务，CovScript 还兼容存量系统。除了主流版本兼容 Windows 7 64bit，还可定制兼容 Windows XP SP2。

2017 年至 2022 年是 CovScript 快速发展的五年，这五年使 CovScript 具备了比肩成熟编程语言的非常成熟的生态。2023 年后，CovScript 开始计划向现代编程语言和前沿应用领域进军，先是完善了第四代语言标准（CovScript 4，或称为 ECS），后是验证了 CovTorch 机器学习框架的可行性。未来，CovScript 将重点着力于建设 LLMOps 流程中的关键基础设施，为前沿应用赋能。

3.5 DeepLang

项目分类	语言类、免费、开源（MIT）、通用、接受社区贡献
语言类别	通用编程语言
工具类别	一般编译工具
应用领域	通用
主页	https://deeplang.org/
仓库	https://github.com/deeplang-org/deeplang

3.5.1 简介

DeepLang 是一种专为资源受限场景设计的编程语言，采用静态类型和强类型特性，语法风格参考 C-style 设计，并支持过程式、逻辑式和函数式的混合编程范式。面向物联网（IoT）应用，DeepLang 语言具备鲜明的内存安全特性，其设计借鉴了 Rust 的安全机制，并针对资源受限场景的特点选择了更合适的编译与执行模式。

DeepLang 的软件体系包括编译器 Deepc 和虚拟机 DeepVM。Deepc 由 OCaml 开发，实现了多阶段的代码处理流程：首先通过语法解析器（parser）生成语法树，然后使用遍历器（walker）多次遍历语法树，构建符号表。转换模块（conversion）根据符号表将语法树转译为 ANF IR（行政范式中间表示），最后通过 codegen 模块将 ANF IR 转换为 WASM 字节码。目前，Deepc 仅支持类型检查器（type checker），尚未实现类型推断器（type infer），因此所有 DeepLang 源码必须显式标注类型，否则会被视为语法错误。DeepVM 由 C 语言开发，支持 WASM 1.0，具备字节码加载、内存管理、解释执行和 FFI 机制等功能。

目前，DeepLang 团队主要由来自江南大学、帝国理工学院、浙江大学以及中国科学技术大学的硕士和博士研究生组成，专注于研究资源受限场景下语言特性的设计。由于团队精力和资源有限，DeepLang 现阶段尚未具备任何商用落地能力。

DeepLang 的 ADT 特性示例：

```
// Some top-level declarations
type Shape [
    Rectangle(width : U32, height : U32),
    Circle(radius : U32),
    Nothing
]

type ColoredPoint {
    as position : Point,
    color : Color
}

fun main (x: Char) {
    // Some more top-level declarations
    let tsr: F64 = 2.72;
    let shape: Shape = Circle(12);
    let point: Point = ColoredPoint {
        position : mut a,
                           color : ColoredPoint {
            position : itmakesnosense,
            color : butsyntacticallycorrect
        }
    };
}
```

DeepLang 的接口特性示例：

```
interface Foo {
    fun foo(x: Int, y: Int) -> ();
    fun bar(x: Int, y: Int)
        -> Bool;
}
interface Bar extends Foo, Bar {
    fun foo(x: Int, y: Int);
    fun bar(x: Int, y: Int)
        -> Bool;
}
impl Foo for Baz {
    fun foo(x: Int, y: Int) {
        print("bla");
    }
}
type Duck [ BaseDuck ]
impl Quack for Duck {
         fun quack() -> () {
        print("quaaaack");
    }
}
```

```
type Bird [ BaseBird ]
impl Quack for Bird {
    fun quack() -> () {
        print("bird quaaaack");
    }
}
fun sound (animal: Quack) -> () {
    animal.quack();
}
fun main() -> () {
    let duck: Duck = Duck();
    let bird: Bird = Bird();
    // type checking pass
    sound(duck); // quaaaak
    sound(bird); // bird quaaaak
}
```

DeepLang 的模式匹配特性示例：

```
fun main() {
    match(x) {
        _ => { return 0; }
        a : Bool => { return 1; }
        Nothing() => { return 2; }
        Some(Any(y)) => { return 3; }
        () => { return 4; }
        mut a => { return 5; }
        (a, b: Bool, (c, d), e: Char): (F32, Bool, (I32, I32), Char)
            =>{ return 6; }
        7 => { return 7; }
        Point { x : (7: I32), y : Point { x : _ } } => { return 8; }
        Point { x : 7, y : Point { x : _ } } : Point as p
            => { return 9; }
    }
}
```

3.6 GödelScript

项目分类	语言类、免费、开源（Apache-2.0）、专用、接受社区贡献
语言类别	领域专用语言、声明式语言
工具类别	编译工具、代码生成、解释器
应用领域	行业应用（代码静态分析、数据库）
主页	https://github.com/codefuse-ai/CodeFuse-Query/blob/main/godel-script/README.md
仓库	https://github.com/codefuse-ai/CodeFuse-Query/tree/main/godel-script

3.6.1 简介

GödelScript 设计于原蚂蚁集团 CodeInsight 团队：陈欣予、范刚、傅先进、梁义南、李皓琨、李世杰、时清凯、王文洋、肖枭、周金果、甄羿等成员（首字母排序）。

GödelScript 版的 Hello world 程序代码如下：

```
@output
pub fn hello(greeting: string) -> bool {
    return greeting = "hello world!"
}
```

GödelScript 是 CodeQuery 用于查询和数据处理的领域专用语言 (DSL)。底层引擎为 Soufflé Datalog。GödelScript 使用了类 Rust 的语法，提供了严格的类型检查、方便快捷的类型推导、智能友好的错误提示信息，使用户能够快速上手。

GödelScript 编译器主要应用场景为：

- 面向用户编写简单或复杂查询，提供更便捷的写法，提高编写查询的效率；
- 提供严格类型检查与类型推导，给予更智能的代码修改提示；
- 提供严格的 ungrounded(未赋值/未绑定) 检测，避免触发 Soufflé Ungrounded Error；
- Language Server 以及 IDE Extension 支持。

由于是基于 Datalog 的 DSL(领域专用语言)，该语言提供了一些具有 Datalog 特征的特性，如：

```
// create schema Person
schema Person {
    name: string,
    age: int,
}

impl Person {
    // define universal set of Person in special method __all__
    pub fn __all__() -> *Person {
        yield Person { name: "John", age: 18 };
    }
    pub fn getName(self) -> string {
        return self.name;
    }
    pub fn getAge(self) -> int {
        return self.age;
    }
}
// create schema Student extends Person
// all methods except __all__ are inherited
schema Student extends Person {}
impl Student {
    pub fn __all__() -> *Student {
        yield Student { name: "Johnson", age: 18 };
    }
    // getName inherited, also can be overridden
    // getAge inherited, also can be overridden
    pub fn getAge(self) -> string {
        return "age: " + (self.age + 1).to_string();
    }
}
```

而获取查询结果的函数可以写为：

```
@output
pub fn student_name(name: string) -> bool {
    for (stu in Student()) {
        return name = stu.getName();
    }
}
```

或者使用 SQL-like 语法：

```
query student_name from
    stu in Student()
select name = std.getName()
```

3.7 HVML

项目分类	语言类、免费、开源（多种许可证方式）、通用、接受社区贡献
语言类别	通用编程语言
工具类别	一般编译工具、运行时环境
应用领域	通用
主页	https://hvml.fmsoft.cn/
仓库	https://github.com/HVML

3.7.1 简介

HVML 是 Hybrid Virtual Markup Language（混合虚拟标记语言）的缩写。它通过标记语言的方式来组织呈现代码。Virtual 表示通过赋予标记语言编程能力，使得该语言成为一种抽象化后的虚拟标记语言。Hybrid 表示混合，它能够通过胶水的方式组织各种不同的语言或者程序。

HVML 编程语言于 2020 年 7 月公开了第一份规范草案；2021 年 7 月开始了 HVML 解释器的开发；2022 年 5 月 30 日，完成了 HVML 图形渲染器 xGUI Pro 的初步版本；2022 年 7 月 3 日，HVML 1.0 解释器 PurC、渲染器 xGUI Pro 趋于稳定，我们公开了 HVML 相关的所有源代码仓库（或软件包）；2023 年 12 月，启动了 HVML 渲染器 xGUI 的开发工作，xGUI 将采用自研的渲染引擎，将于 2024 年 12 月公开。

HVML 的基本设计目标是，在已有的以 C/C++，Python 等编程语言构造的原生运行时环境中，利用现代 Web 前端技术(HTML/SVG、DOM、CSS 等)快速开发图形用户界面程序，而不需要借助额外的浏览器或者 JavaScript 引擎。

描述性是 HVML 的特点。描述性的语言不但能够方便开发者理解和撰写代码，也适合 AI 程序进行学习和代码生成。

```
<!--
    $SYS.locale returns the current system locale such as `en_US` or `zh_CN`
    $STR.substr returns a substring of the given string.
-->
<hvml target="html" lang="$STR.substr($SYS.locale, 0, 2)">

    $STREAM.stdout.writelines('Start of `Hello, world!`')
```

```
    <body>

        <!-- 'test' element checks whether the system locale starts with
`zh` -->
        <test with = $STR.starts_with($SYS.locale, 'zh') >

            <h1>我的第一个 HVML 程序</h1>
            <p>世界，您好！ </p>

            <!-- If the system locale does not start with `zh` -->
            <differ>
                <h1>My First HVML Program</h1>
                <p>Hello, world!</p>
            </differ>
        </test>

    </body>

    $STREAM.stdout.writelines('End of `Hello, world!`')

</hvml>
```

HVML 可以非常方便的与其他程序进行数据交互，比如和高精度计算程序 `bc` 交互实现图形界面版的高精度计算器：

图 3.7-1

以及可以内嵌 python 代码，与 python 程序进行数据交互，处理和显示：

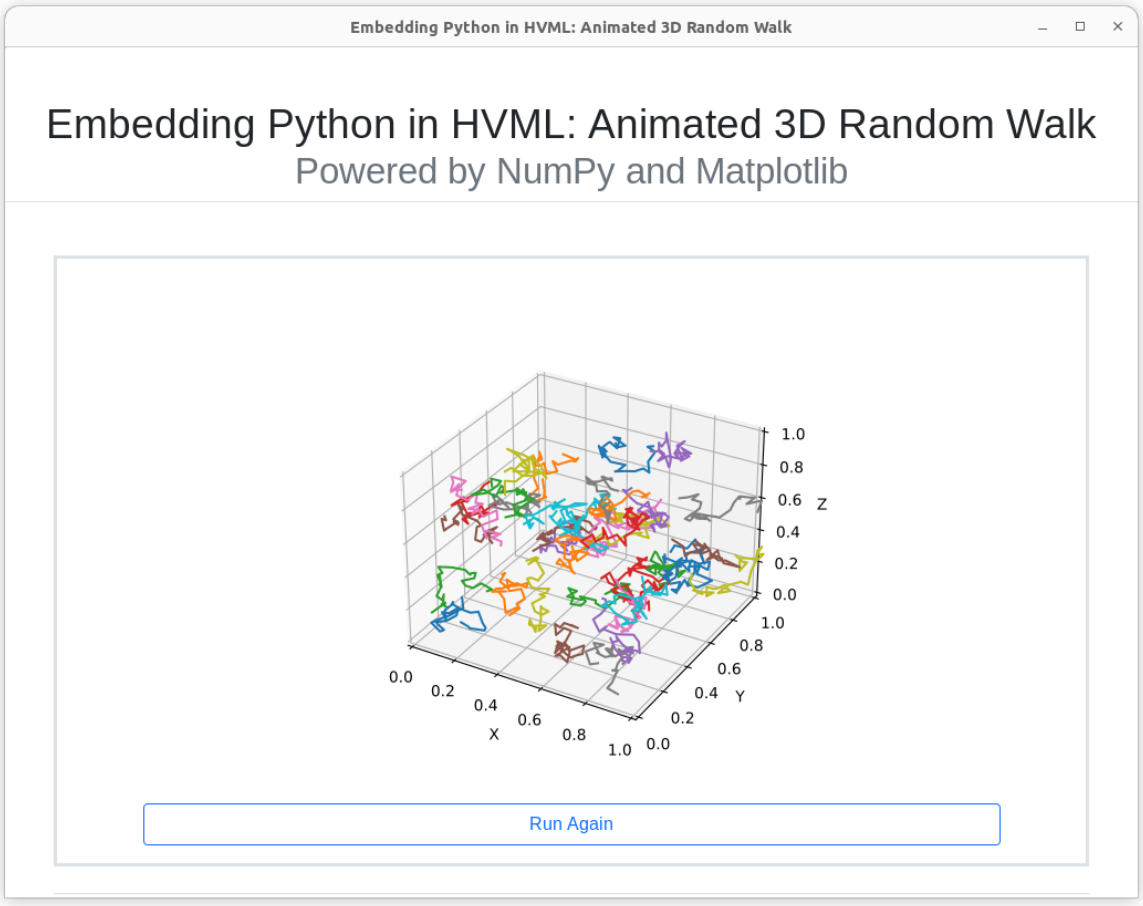


图 3.7-2

作为一个标记语言，标记符号的引入使得代码字符数量比其他语言要多，但是好处是对代码的组织会显得更加清晰。这是因为它的另一个设计目标就是借助自动化图形化低代码的开发工具来进行程序开发，同时清晰的组织也方便接入 AI 应用。

3.8 金鱼 Scheme

项目分类	语言类、免费、开源（Apache-2.0）、通用、接受社区贡献
语言类别	通用编程语言、函数式语言
工具类别	解释器
应用领域	行业应用（教育、科研）
主页	https://gitee.com/LiiiLabs/goldfish
仓库	https://gitee.com/LiiiLabs/goldfish

3.8.1 简介

金鱼 Scheme 是一个 Scheme 解释器，具有以下特性：

- 兼容 R7RS-small 标准
- 提供类似 Python 的标准库
- 小巧且快速

金鱼 Scheme 的设计目标是让 Scheme 和 Python 一样易用且实用。

金鱼 Scheme 是应用场景驱动的编程语言项目。由于墨干理工套件所使用的 S7 Scheme 并不能满足墨干理工套件的长远发展，我们发起了金鱼 Scheme 这个项目，以满足墨干内置的 7 万 6 千行历史 Scheme 代码的长远维护，另外，墨干中以 Python 语言实现的插件，比如 Gnuplot 绘图插件，我们也从 Python 实现切换到了金鱼 Scheme 实现。

金鱼 Scheme 是采用文学编程方式实现的。我们认为：大模型时代，以代码为中心的传统编程范式，会切换到以文档为中心的文学编程范式，编程的门槛会持续降低。文学编程作为一种历史悠久的编程范式会在大模型时代经历一场文艺复兴。

金鱼 Scheme 的文档是中文优先的，因为发起者的母语是中文。除文档之外的社区交流，比如代码提交信息、代码合并请求标题和内容、社区开发者的飞书群均鼓励使用英文。

3.9 KCL

项目分类	语言类、免费、开源（Apache-2.0）、专用、接受社区贡献
语言类别	领域专用语言、声明式语言
工具类别	一般编译工具
应用领域	行业应用（云原生、数据工程、平台工程等）
主页	https://kcl-lang.io/
仓库	https://github.com/kcl-lang/kcl

3.9.1 简介

KCL 是一个开源的基于约束的记录及函数语言。KCL 通过成熟的编程语言技术和实践来改进对大量繁杂配置比如云原生场景的编写，致力于构建围绕配置的更好的模块化、扩展性和稳定性，更简单的逻辑编写，以及更快的自动化集成和良好的生态延展性。

自 2022 年 5 开源以来，已被许多全世界各地公司主体或个人采用选用并投入生产使用，并在 2023 年 9 月正式捐赠给 CNCF 基金会。

您可以将 KCL 用于：

- 生成静态配置数据如 JSON，YAML 等，或者与已有的数据进行集成；
- 使用 schema 对配置数据进行建模并减少配置数据中的样板文件；
- 为配置数据定义带有规则约束的 schema 并对数据进行自动验证；
- 通过梯度自动化方案无副作用地组织、简化、统一和管理庞大的配置；
- 通过分块编写配置数据可扩展地管理庞大的配置。

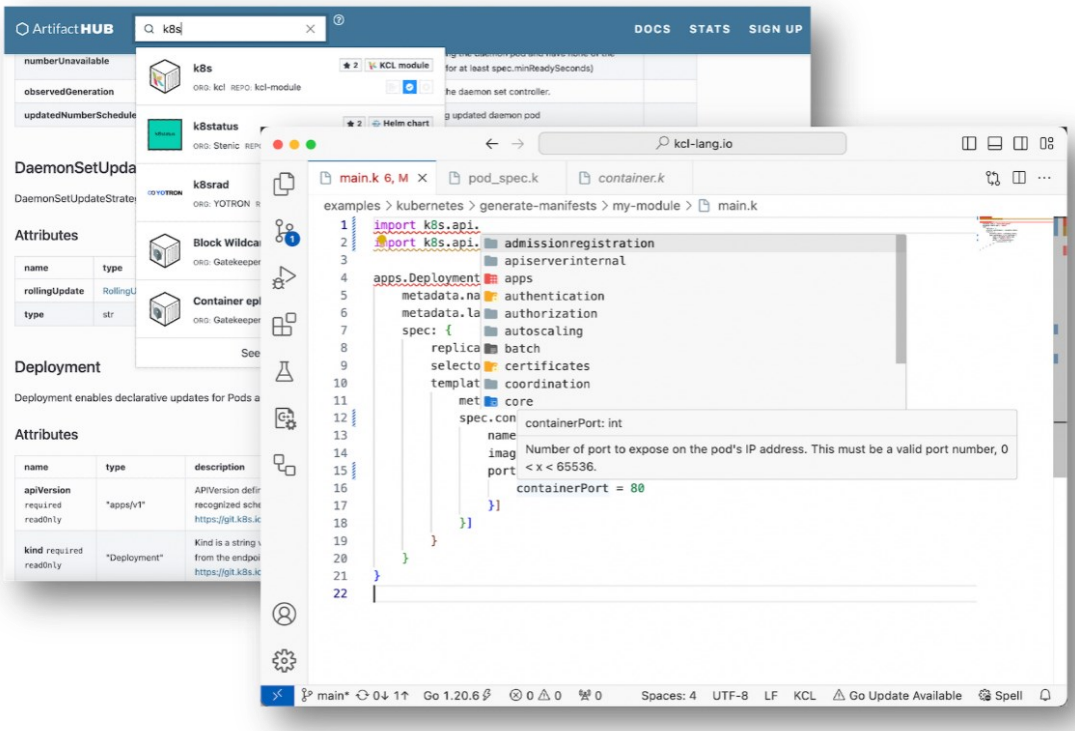

Artifact HUB
k8s
DOCS
STATS
SIGN UP
k8s
k8status
k8srad
DaemonSetUpda
Attributes
Deployment
Attributes
kcl-lang.io
main.k 6, M
pod_spec.k
container.k
examples > kubernetes > generate-manifests > my-module > main.k
import k8s.api.
import k8s.api.
apps.Deployment
admissionregistration
apiserverinternal
apps
authentication
authorization
autoscaling
batch
certificates
coordination
core
containerPort: int
Number of port to expose on the pod's IP address. This must be a valid port number, 0 < x < 65536.
containerPort = 80
Go 1.20.6
Spaces: 4
UTF-8
LF
KCL
Go Update Available
Spell

图 3.9-1

3.10 Koral 语言

项目分类	语言类、免费、开源（MIT）、通用、接受社区贡献
语言类别	通用编程语言
工具类别	一般编译工具
应用领域	通用
仓库	https://github.com/kulics/koral

3.10.1 简介

Koral 语言是面向应用领域的开源编程语言。具有静态类型、内存托管、多范式的特点。现阶段 Koral 语言的主要目标是探索类型系统和语法设计，还不具备任何商用能力，也不承诺任何稳定性。

目前 Koral 语言尝试了几个比较有意思的设计：

- 通过大小写区分的泛型语法；
- 基于分号和块区分的表达式结构语法；
- 可参数化的可变类型限定符。

代码示例 1：

```
type Pair(T1 Any, T2 Any)(left T1, right T2);
let main() = {
    lef a1 Pair(Int, Int) = Pair(1, 2);
    ## a1.left is Int, a1.right is Int
    lef a2 Pair(Bool, Bool) = Pair(true, false);
    ## a2.left is Bool, a2.right is Bool
    lef a3 Pair(Int, String) = Pair(1, "a");
    ## a3.left is Int, a3.right is String
}
```

代码示例 2：

```
let main() = {
    if true or f() then {
        ...
    }
    0
}
```

代码示例 3:

```
type mut Point(x Int, y Int);
let main() = {
    let a mut Point = mut Point(64, 128);
    let b Point = a; ## ok
    printLine(a.x); ## 64
    printLine(b.x); ## 64
    a.x = 128;
    printLine(a.x); ## 128
    printLine(b.x); ## 128
    b.x = 256; ## error
}
```

3.11 洛书（Losu）编程语言

项目分类	语言类、免费、开源（MIT）、通用、接受社区贡献
语言类别	一般编程语言
工具类别	解释器、运行时环境
应用领域	通用、行业应用
主页	https://losu.tech
仓库	https://gitee.com/chen-chaochen/lpk

3.11.1 简介

洛书（Losu：Language of System Units，也称 Easylosu，Losuscript）是一门超轻量级的跨平台脚本语言，针对 IoT 场景下的低资源设备控制与轻业务开发设计，其最低资源需求仅为 RAM ≥ 4KB，code space ≥ 32 KB，旨在提供轻量级、动态化的脚本编程能力，提高项目灵活性、扩展性和定制功能。

洛书是一门创新性的编程语言，拥有简洁、高效且可靠的设计与实现：

- 洛书主体语法采用类 Python 风格，规则简洁，上手迅速：

```
# HelloWorld for Losu-Language
def sayHello(msg):
    print(msg)
var s= "Hello World!"
sayHello(s)
```

- 洛书支持原生支持 JSON 格式，高效的数据交换与配置体验：

```
# 洛书支持 JSON 格式
var menu = {
    "menu": {
        "id": "file",
```

```
        "value": "File",
        "popup": [
           { "value": "New", "onclick": "CreateNewDoc()" },
           { "value": "Open", "onclick": "OpenDoc()" },
           { "value": "Close", "onclick": "CloseDoc()"},
           { "value": "Rm", "onclick": "RmDoc()"},
           { "value": "Move", "onclick": "MoveDoc()"}
        ]
     }
  }
```

- 洛书支持多范式编程、支持闭包、高阶函数、鸭子类型、运算符重载等高级特性：

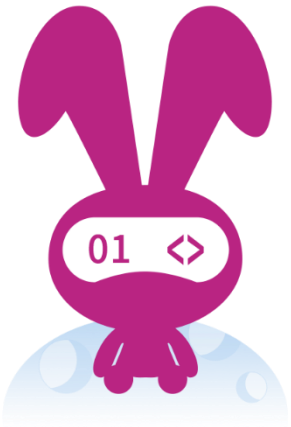
```
# 闭包
def outer_func(val):
   return def():
      print(val)
var f = outer_func(0)
f()
```

- 洛书拥有良好的并发支持：洛书抛弃了传统的 GIL 方案，自行设计实现了 C(协程)/T(线程)/P(进程) 三级调度模型。其中，洛书协程为原生数据结构，可以在裸机环境下使用：

```
# 协程
import 'math'
def task(fname, f):
   for i in 1,5:
      print(fname, ':', i, '=', f(i))
         yield
var t1 = async(task, 't1', sqrt)
var t2 = async(task, 't2', pow)
while await (t1,t2):
   pass
```

洛书已经于 2023 年 6 月发布了最小可行化版本（MVP），实现了完整的语法设计、编译器、虚拟机、核心库、拓展库，并基于自身实现了包管理器、Playground 等工具，初步成为一门具备实用性的新兴编程语言。同时，洛书预计在 2025 年初发布首个最小可市场化产品（MMP）与相应的 SDK 组件。

3.12 MoonBit



项目分类	语言类、免费、开源、通用、接受社区贡献
语言类别	通用编程语言
工具类别	一般编译工具
应用领域	通用
主页	https://www.moonbitlang.cn/
仓库	https://github.com/moonbitlang/core

3.12.1 简介

MoonBit 是一个语法与 Rust 类似的编程语言（带有 GC 支持），带有现代化的工具链及多后端支持。MoonBit 版的 hello world 程序代码如下：

```
fn main {
  println("Hello World")
}
```

主要优势：

- 优秀的编译与构建速度；
- 简单但实用、面向数据的语言设计；
- 多后端支持，包含 WebAssembly、WebAssembly 使用 GC 提案、JavaScript、C 后端等；
- 生成 WebAssembly 代码体积小，运行速度快；生成 C 后端运行高效。

特性：

- 云原生开发：MoonBit 提供云原生 IDE，包含所有现代 IDE 的功能，可以在不依赖任何后端的情况下在浏览器中进行项目开发、运行、测试、调试等；
- 全链条设计：MoonBit 设计时便规划实现完整生态链，包括 IDE、编译器、语言服务器

等组件，及开发、测试、调试、发布等流程；

- 面向 AI 设计：MoonBit 设计时便考虑与 AI 的结合，语言设计便于 AI 生成。所有函数、方法均在顶层定义，具有完整类型声明，并且采用结构化 Trait，如：

```
pub(open) trait Animal {
  speak(Self) -> Unit
}

struct Dog { }

// implements Animal
pub fn speak(self : Dog) -> Unit {
  println("Bark")
}

let animals : Array[Animal] = [ Dog::{} ]
```

- 面向应用开发者设计：MoonBit 原生支持 JSON 语法，如：

```
let data : Json = {
    "object" : { "key": 1, "value" : "v" },
    "array" : [ 1, true ]
}
```

项目展示：

- 使用 MoonBit 与 Wasm4 开发游戏，运行在浏览器与 ESP-C6 单片机中：

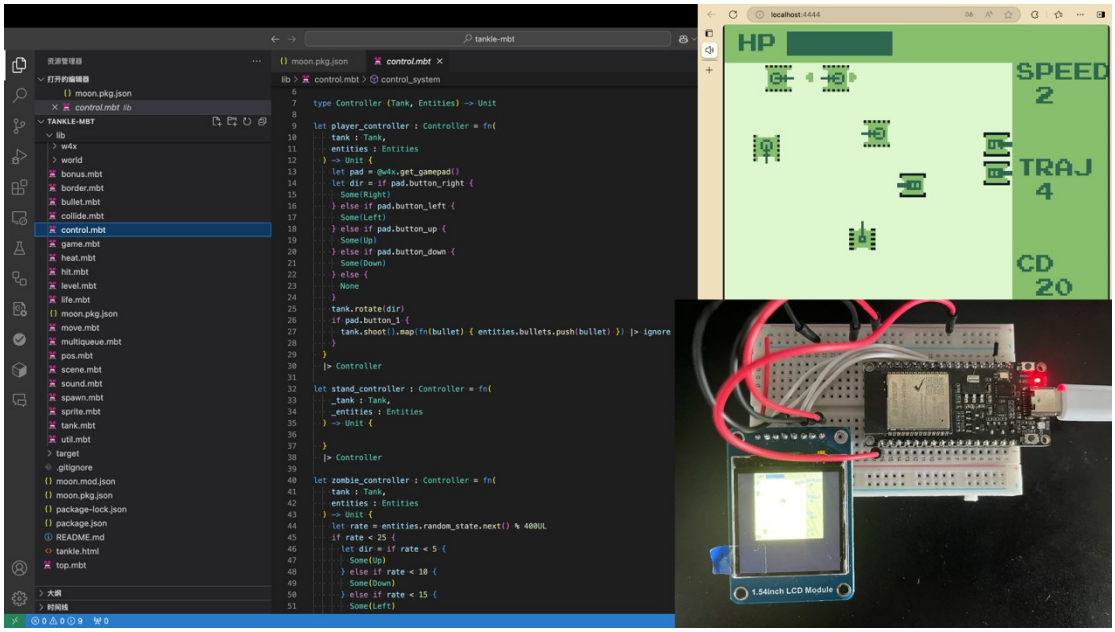


图 3.12-1

- 使用 MoonBit 开发的网页应用：

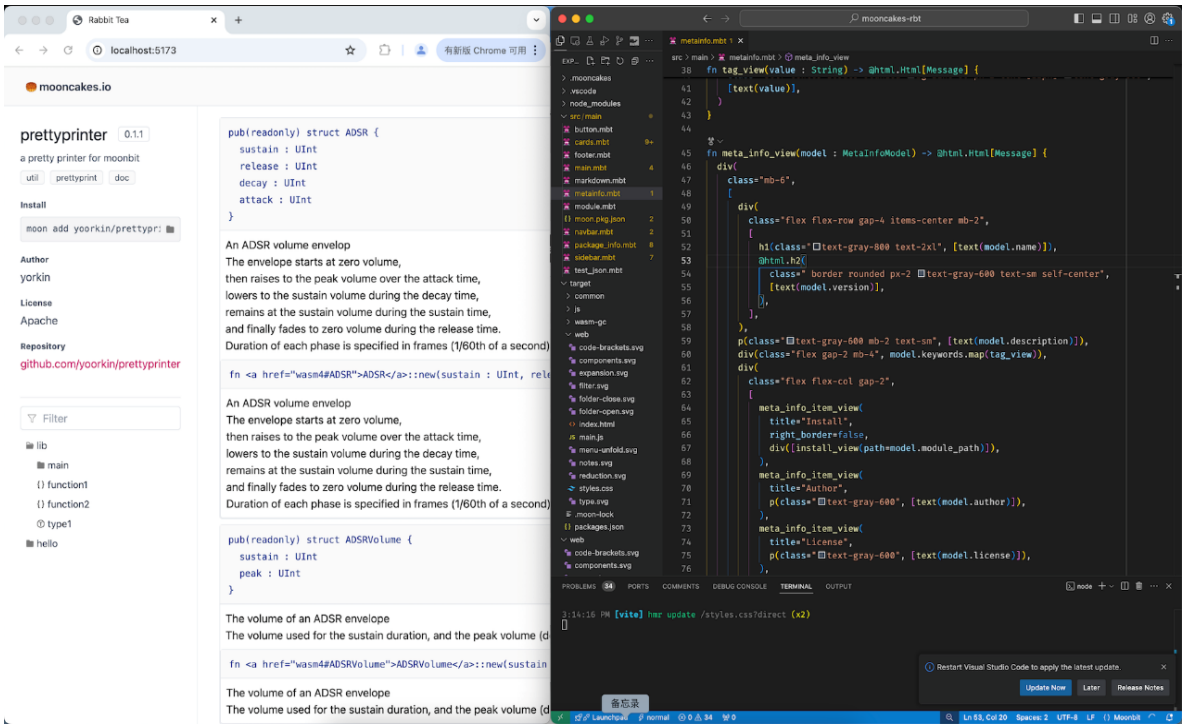


图 3.12-2

3.13 Nasal-Interpreter

项目分类	工具类、免费、开源（GPLv2）、通用、接受社区贡献
工具类别	脚本语言、解释器
应用领域	通用、行业应用
主页	https://www.fgprc.org.cn/nasal_interpreter.html
仓库	https://github.com/ValKmjolnir/Nasal-Interpreter

3.13.1 简介

Nasal 是一款语法与 ECMAscript 相似的脚本语言，设计者为 Andy Ross。后来被引入著名的开源飞行模拟器 FlightGear 中，作为 FlightGear 机模开发专用的脚本语言。Nasal 版的 hello world 程序代码如下：

```
print("hello world!");
```

由于在 FlightGear 中，使用内嵌的 Nasal 控制台窗口进行调试很不方便，仅仅是想检查语法错误，也得花大量时间打开软件等待加载后进行调试。

所以一个全新的 Nasal 解释器诞生了。项目的初衷是帮助开发者检查语法错误，甚至是运行时的错误。在近期的迭代中，Nasal-Interpreter 还支持了 REPL 解释器，跨平台 subprocess，以及更加详细的错误 trace back 信息。

Nasal 的基础数据类型也非常简单，复杂的数据类型可以通过基础类型的组合来实现：

```
var this_is_number = 0.0;
var this_is_string = "i am string";
var this_is_vector = [0, "i am vector", ["another vector"]];
var this_is_hash = {
    field_name: "field_value",
    parents: [{}]
};
```

Nasal 的函数实际上是 Lambda，可以作为数据进行传递：

```
var this_is_function = func(a, b) {
    return a + b;
}
var hash = {
    f: this_is_function,
    example: func(a, b) {
        # `me` acts like `this` in other languages
        return me.f(a, b);
    }
};
print(hash.f(1, 2), "\n"); # expect 3
print(hash.example(2, 4), "\n"); # expect 6
```

Nasal 使用一个比较特别的机制来模拟继承：

```
var parent = {
    prt: func { print("in parent function\n"); }
}
var child = {
    parents: [parent]
}
child.prt(); # expect "in parent function\n"
```

Nasal 还使用两种特别的循环语法来方便脚本的编写：

```
forindex(var i; [0, 0, 0]) {
    print(i); # expect 012
}
foreach(var i; ['foo', 'bar']) {
    print(i, " "); # expect foo bar
}
```

3.14 NASL

项目分类	语言类、商用(免费试用无时长限制)、闭源、专用、不接受社区贡献
语言类别	领域专用语言、可扩展语言
工具类别	代码生成、增量编译器
应用领域	行业应用（低代码平台）
主页	https://nasl.codewave.163.com/

3.14.1 简介

NASL，全称 Next Application Specific Language，是网易数帆 CodeWave 智能开发平台(https://sf.163.com/product/lcap?productId=neteasecloud)用于描述 Web 应用的领域特定语言。它主要包含两部分：基础语言和 Web 应用特定领域（如数据源、数据查询、页面、流程、权限等）的子语言集合。NASL 最主要的特点是使用 CodeWave 智能开发平台的可视化编辑器来统一设计 Web 应用的页面、业务逻辑、数据、流程等方方面面，并辅有静态检查、全栈调试、AIGC 代码生成、多人协作等功能：

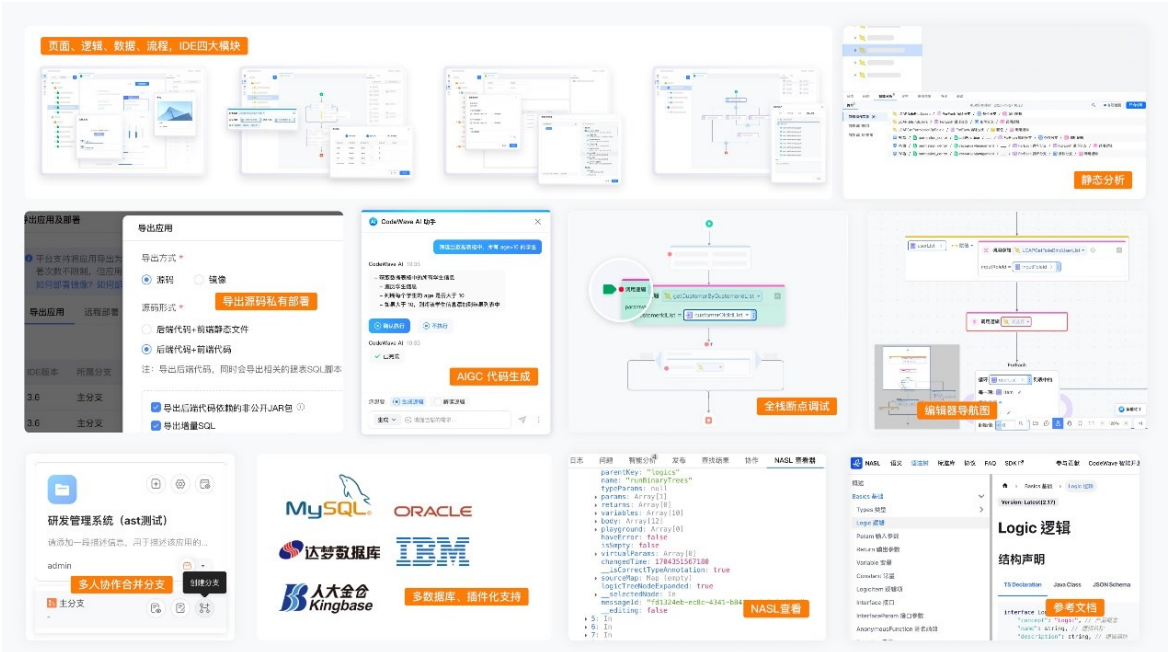


图 3.14-1

对搭建 Web 应用来说，NASL 及其配套设施开箱即用，学习门槛低，开发成本少：开发者不

需要再学习多门框架、语言（如前端 TypeScript、Vue，后端 Java、Spring），也不需要在他们之间互转数据。

NASL 及其配套设施的整体架构图如下：

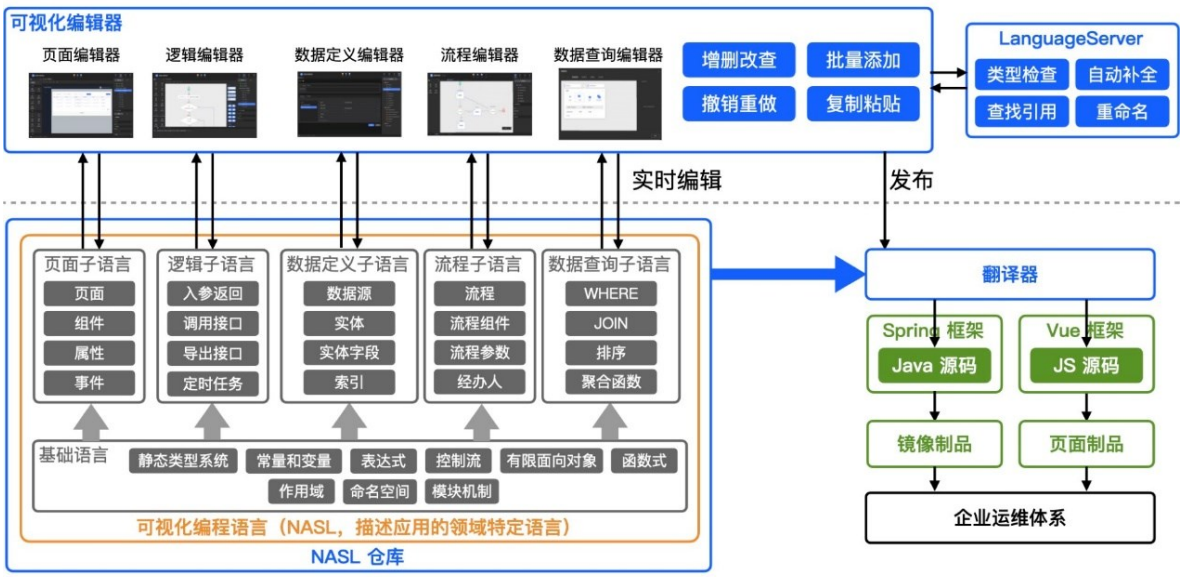


图 3.14-2

2024 年项目依旧在 IDE 侧扩充了较多功能，在 NASL 侧则新增了文本语法、异常处理、break、continue 等能力。

下面分别介绍基础语言、子语言、配套设施。

基础语言：NASL 基础语言融合了面向对象、函数式等编程范式中常见的语言特性，有着和大多数通用计算机编程语言一样的表达能力：

- 使用静态类型系统，支持常用的原子类型、复合类型、集合类型和数据元类型等。
- 提供了联合类型（union types）和匹配（match）表达式。
- 支持逻辑（函数）定义，逻辑里可使用常见的 if、while、foreach 等控制流和 lambda 表达式。
- 支持命名空间、模块化和依赖机制。
- 提供常用的内置函数标准库。

NASL 利用可视化对复杂的语言特性做了屏蔽和简化，大大降低了用户的学习门槛，符合低代码群体的用户画像。

子语言：NASL 子语言是在基础语言之上，吸收了 Web 应用各子领域的传统编程语言和框架的主要特征而设定的 DSL，其中：

- 数据定义子语言用于表达数据库、表、字段和索引等相关概念。
- 数据查询子语言用于表达筛选、排序、分页和聚合等数据查询场景。

- 页面子语言主要用于表达页面布局、页面交互和页面样式等场景。
- 流程子语言主要用于表达手动任务、自动任务、排他网关等流程领域的相关概念。

各子语言并非互相独立、拼凑而成，而是建立在基础语言之上，较为统一，例如：

- 前端、服务端、实体均使用统一的类型定义。
- 前端页面逻辑、服务端逻辑、流程逻辑可使用统一的表达式、语句、内置函数标准库。
- 前端调用服务端逻辑，逻辑调用接口，流程跳转页面等功能屏蔽了底层细节，用户无感。

配套设施：

- Language Server：包含类型检测、类型推断、跳转定义、自动补全等能力，减少编程出错概率和提高编程效率。
- Debugger：包含 breakpoint、step into、step over、resume、evaluation 等能力。
- 代码仓库：用于实时保存用户构建应用所产生的 NASL 代码，并满足高性能、高可用、高可靠等特性。
- Generator：NASL 语义编译器。低代码平台借助于 Generator，将 NASL 语言编译为 Java、JavaScript 等通用语言，在借助底层通用语言的运行时设施如 JVM，将 NASL 语言运行在计算机上。
- Upgrader：用于 NASL 语言在版本迭代过程中产生的一些兼容性问题处理。

库与依赖、编译器架构等其他方面详见轻舟低代码技术白皮书：
http://nasl.codewave.163.com/%E6%8A%80%E6%9C%AF%E7%99%BD%E7%9A%AE%E4%B9%A6V1.0-1118.pdf。

3.15 PikaPython

项目分类	免费、开源（MIT）、通用、接受社区贡献
工具类别	解释器、运行时环境
应用领域	行业应用
主页	https://pikapython.com
仓库	https://gitee.com/Lyon1998/pikapython

3.15.1 简介

PikaPython 是一个完全重写的超轻量级 python 引擎，零依赖，零配置，极易部署和扩展，具有大量的中文文档和视频资料。项目架构如下图：

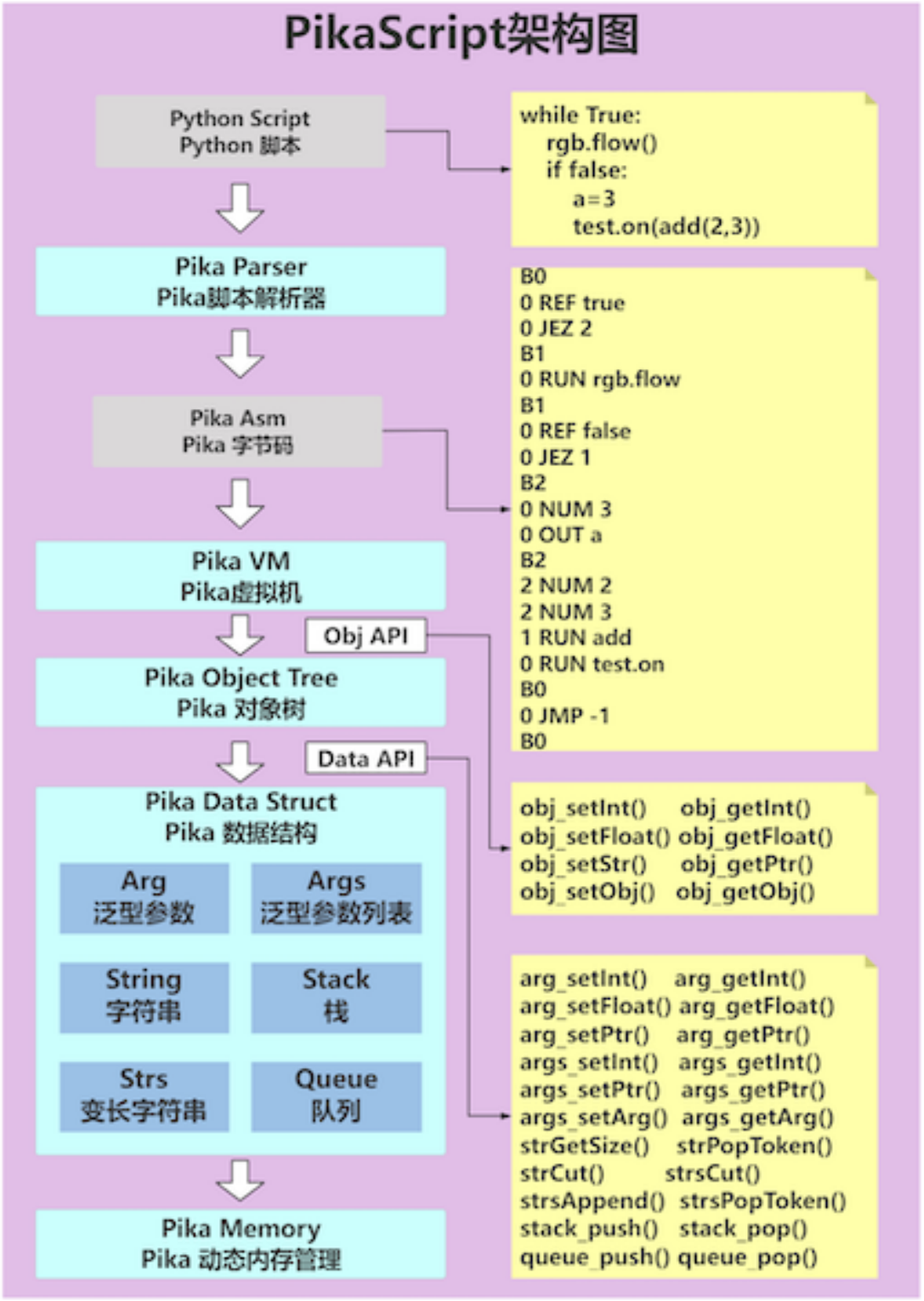


图 3.15-1

- 运行环境:支持裸机运行,可运行于**RAM ≥ 4kB,FLASH ≥ 64kB**的mcu中,如stm32g030,stm32f103c8t6，esp8266；
- 开发环境：支持 Keil、IAR、rt-thread studio、segger embedded studio 等 IDE 开发；支持 CMake、makeFile、Scons 等构建工具；零依赖，零配置，开箱即用，极

易集成进已有的 C 工程；极易拓展自定义的 C 原生函数；支持跨平台，可在 linux 环境开发内核；支持串口下载 Python 脚本，如图：

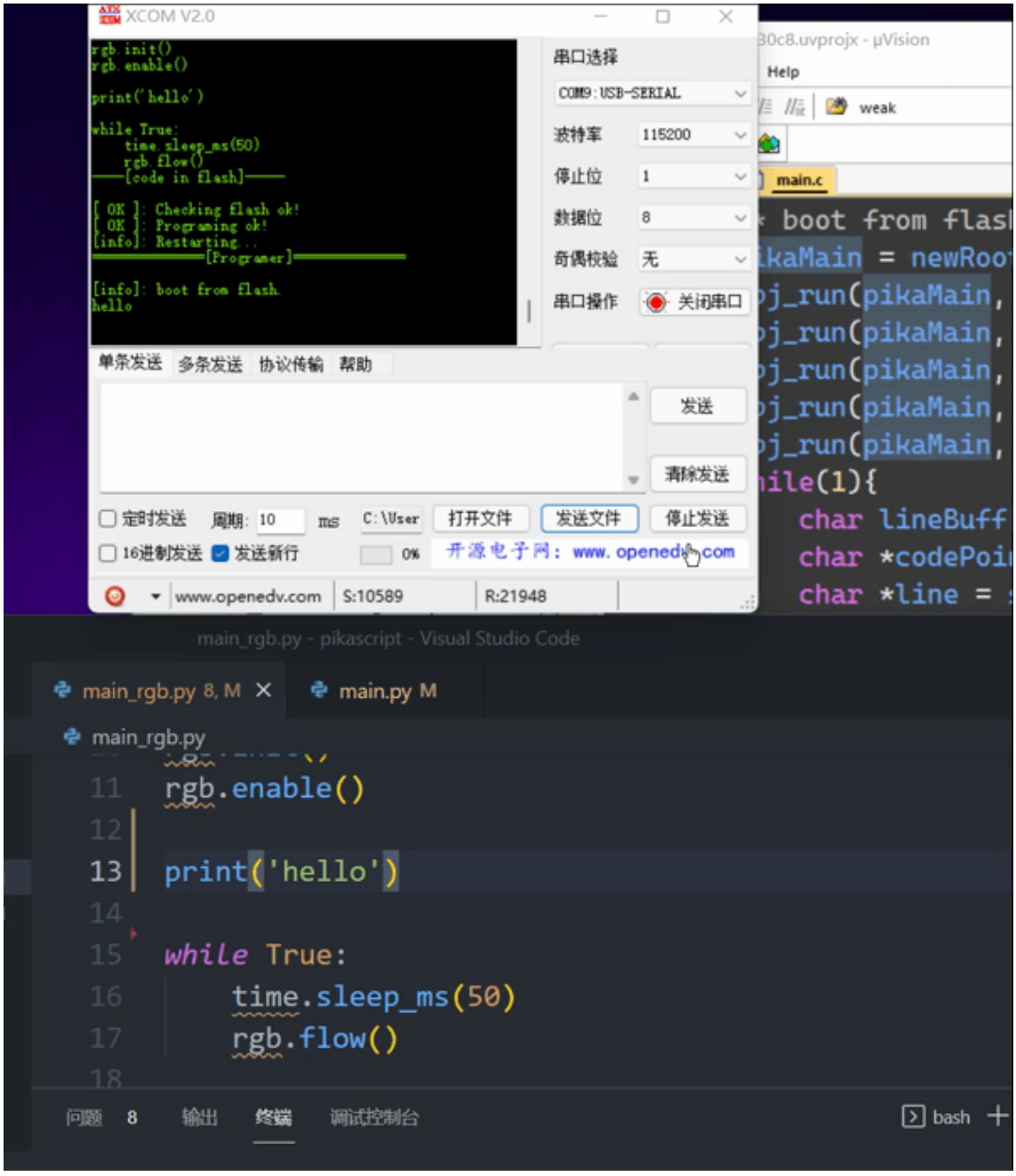


图 3.15-2

- 语法特性：使用 python3 标准语法的子集；在编译时支持 python 类和方法定义，完整支持封装、继承、多态、模块功能 - 基于 **Pika 预编译器**；在运行时支持 python 方法调用、变量定义、对象构造、对象释放、控制流(if\while) - 基于 **Pika 运行时内核**。
- 源码规范：注重源码可读性，命名规范，标准统一，完全不使用宏，几乎不使用全局变量。完整的 googletest 单元测试。

3.16 青语言

项目分类	语言类、免费、开源（木兰宽松协议 V2.0）、通用、接受社区贡献
语言类别	一般编程语言
工具类别	解释器、IDE
应用领域	通用
主页	https://qingyuyan.cn/
仓库	https://gitee.com/NjinN/Qing

3.16.1 简介

青语言是一门完全基于中文语言习惯打造的编程语言，主要面向青少年、儿童和非专业人士。主要设计构成如下：

- 语言核心参考 Lisp 语言。 Lisp 被称为实现编程语言的语言，其极简的语言内核，非常便于实现。这样可以使得青语言的核心语言实现十分简洁，方便开源开发者参与和推动语言核心的发展。
- 语法上参考 JavaScript 语言。 JS 编程语言语法非常简单，其最初设计也是 Lisp 核心，因此实现起来非常容易。对于使用者来说，需要掌握的概念少，可以很容易地学习和使用。
- 使用 C#开发，运行在.Net 平台上。 目前.Net 平台可以说是最开放、跨平台兼容最好的编程语言之一，且本身有良好的语言生态可以借用。基于.Net 平台可以使青语言具备是否良好的跨平台兼容性，同时可以方便地扩展其功能。
- 目前使用动态链接库 DLL 的方式扩展功能。 青语言提供简单的 C#原生功能的封装方法，开发者可以通过参照示例项目，将需要的功能封装成为单个 DLL 文件，可以方便地分享和使用。

青语言提供了解释器、编辑器、安卓 APP，同时支持 Windows、Linux、OSX 兼容，支持 GUI 图形界面编程。

简单时钟示例：

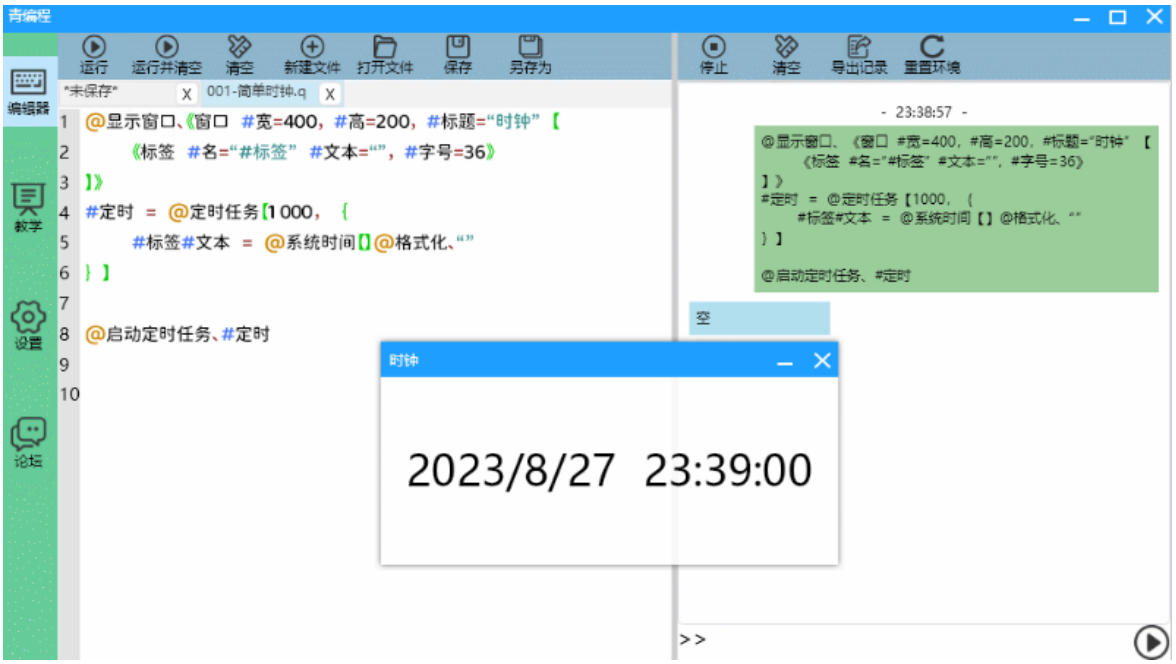


图 3.16-1

AI 图片分类：

图 3.16-2

运行大语言模型：


青编程
运行
运行并清空
清空
新建文件
打开文件
保存
另存为
停止
清空
导出记录
重置环境
编辑器
教学
设置
论坛
未保存
语青对话.q
1 #大语言模型库 = @加载动态库、"./大语言模型库.dll"
2 #模型 = #大语言模型库@创建模型【"E:\\llm_models\\baichuan2-7b-chat.Q8_0.gguf"】
3 #模型@开始对话【】
- 19:01:14 -
#大语言模型库 = @加载动态库、"./大语言模型库.dll"
#模型 = #大语言模型库@创建模型【"E:\\llm_models\\baichuan2-7b-chat.Q8_0.gguf"】
#模型@开始对话【】
正在加载模型，请稍候
初始化完成，开始对话
用户输入:
- 19:05:25 -
什么是室温超导
语青：室温超导是指能够在室温下实现超导电性的材料或系统。这种现象在科学界引起了很大的关注，因为它有可能为许多领域带来革命性的变化，例如能源储存、传输和通信等。你可以进一步了解这个主题的相关资料和研究进展。
用户输入:
- 19:08:11 -
目前有发现室温超导材料吗
语青:是的，目前已经有一些证据表明存在一些具有室温超导电性的材料。这些材料被称为高温超导体（high-temperature superconductors）或氢化物超导体（hydrides）。例如，YBa2Cu3O7-x就是一个典型的高温超导体，它在低温下表现出强烈的超导性，而其临界温度（即从正常导体转变为超导体的温度）已经达到了接近室温的水平。然而，虽然这些材料的研究取得了显著的进展，但目前我们还无法在实际应用中实现真正的室温超导。
用户输入:
>>

图 3.16-3

3.17 狮偶

项目分类	语言类、免费、开源（Apache-2.0）、通用、接受社区贡献
语言类别	一般编程语言、并发编程语言、面向对象语言
工具类别	一般编译工具、解释器、代码生成、运行时环境
应用领域	通用
仓库	https://gitee.com/openblock/openblock

3.17.1 简介

狮偶是开源、面向状态机、图形化、跨平台、IDE 一体的脚本编程语言。狮偶是完全面向业务的编程语言，可以用于构建各种业务系统。不负责技术底层的实现，只负责业务逻辑的描述。IDE 原生提供全场景能力，可以在一个工程里串联整条业务线。

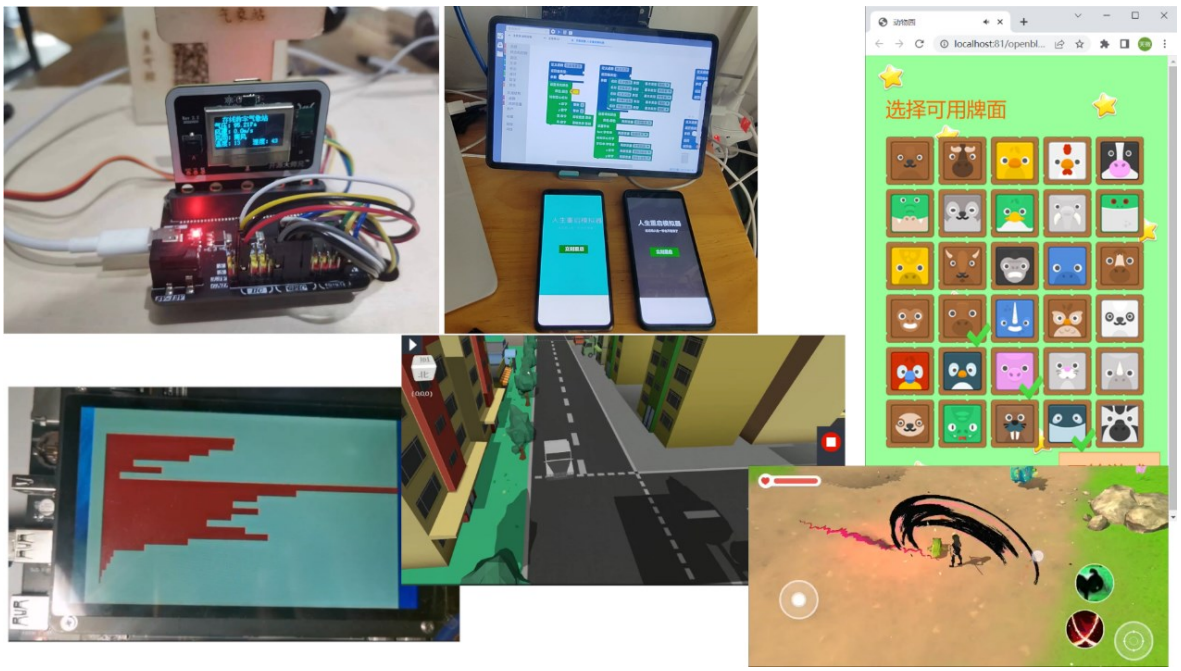


图 3.17-1

狮偶的编译器、解释器、运行时、IDE 全部开源，可以自由修改。狮偶 IDE 集成资源管理、静态数据管理等功能，可以二次开发增加特定领域系统。IDE 提供图形化反馈能力，可以方便业务人员理解业务逻辑。静态数据可以由 Excel 编辑，在编译时随代码一起编译到二进制文件中，降低存储占用，提高运行效率。借助图形化的国际化能力，狮偶编程语言可以在代码完成后应用国际化，实现所有内置和本地库的国际化。

狮偶已经在外企、党政、科研、教育等领域实验性商用。同时在青少年编程教育领域已经获得广泛的应用，支持了两届教育部全国中小学白名单比赛。

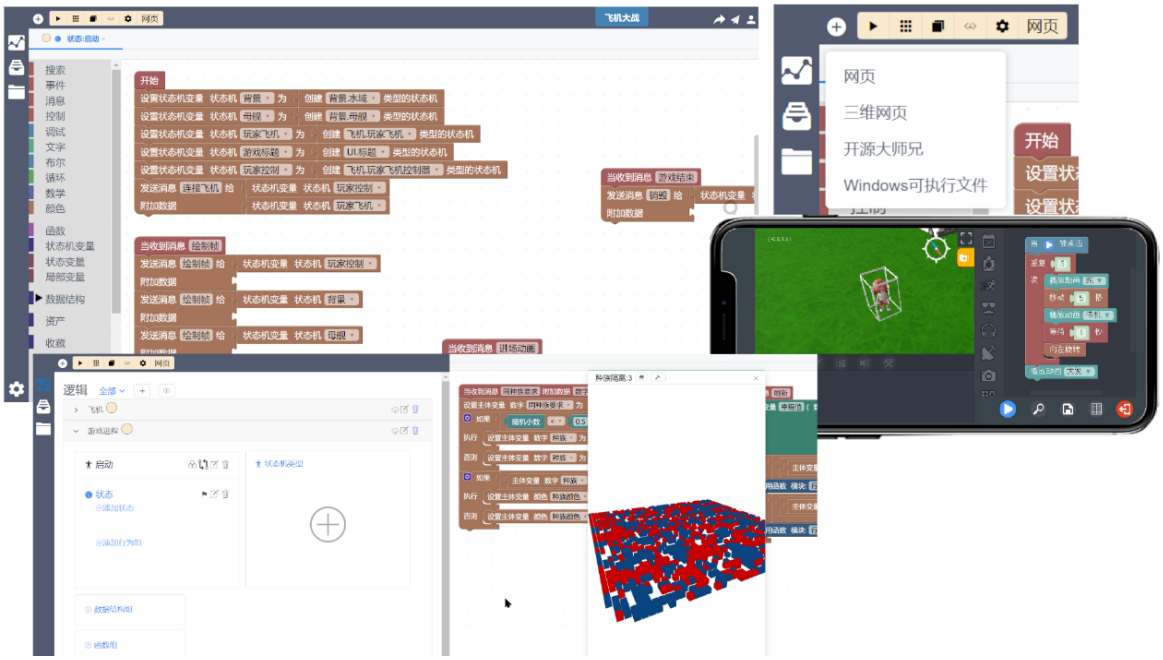


图 3.17-2

狮偶的前端使用 Blockly、Vue 等开源技术构建本身并没有使用前端构建技术，修改源码刷新生效，易于维护和扩展。

编译和连接是完全使用 JavaScript 语言编写，可在浏览器或 Nodejs 中运行。IDE 为纯 Html5 架构，对服务器端无要求，任何 HTTP/HTTPS 服务器都可以运行。业务侧用户不需要配置环境，不需要安装任何软件。主流浏览器均可运行、支持移动端。

狮偶是面向状态机编程的语言，不同的状态可以监听不同的事件、执行不同的业务逻辑，可以很方便的构建各种复杂的业务系统。状态机之间通过异步消息通信，可以充分利用多线程、多进程、分布式等并行技术，实现高并发和高性能。

狮偶是强类型语言，支持自定义数据结构，本地库可支持泛型。函数分为函数、状态机行为、状态行为三种绑定关系。变量分为状态机、状态、局部三个作用域。

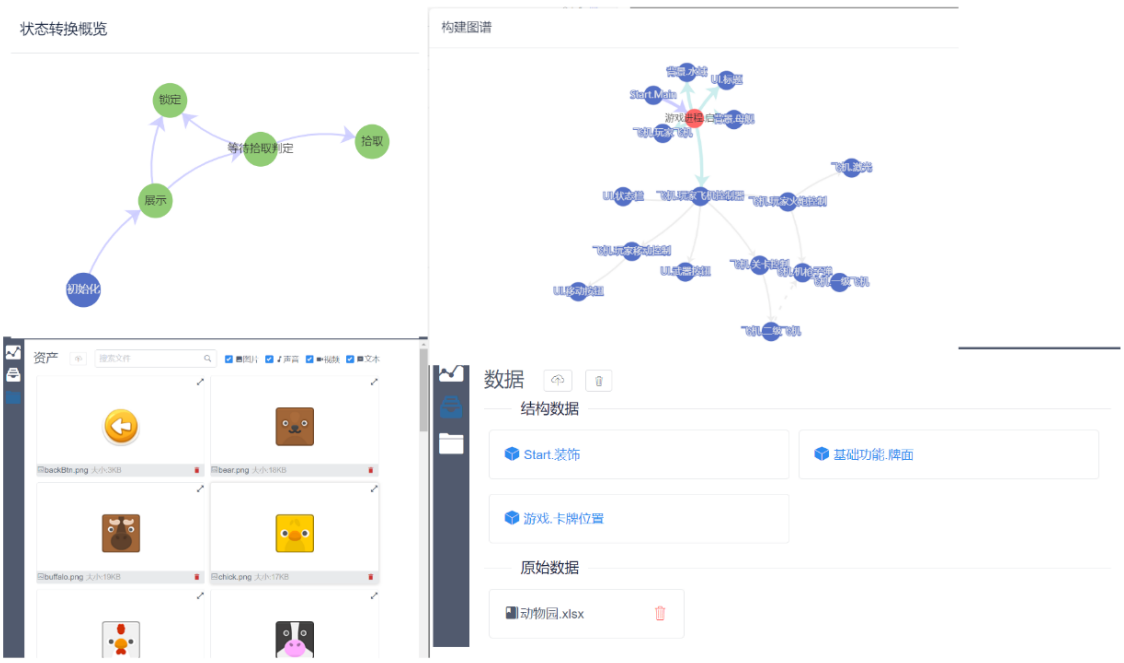


图 3.17-3

通过狮偶 VM 对字节码解释执行，或将字节码二次编译为其他语言代码，实现跨平台和高性能运行。目前主要通过 VM.mjs 在 JS 中解释字节码运行。同时通过 toC.mjs 提供了将字节码编译为 C 代码的功能。由于狮偶大量使用异步逻辑，在可以根据使用场景和技术栈，在技术层控制并发和异步，充分发挥硬件性能，同时也能支持嵌入式、服务器、客户端、网页、可执行文件等运行环境，可以开发 VR、微信小程序、物联网等应用。符合信创要求，支持鸿蒙系统。

狮偶意思是适合摆放在手边激励自己的狮子造型的玩偶。英文名称 roarlang。

3.18 凸语言

项目分类	语言类、免费、开源（AGPL-3.0）、通用、接受社区贡献
语言类别	一般编程语言、命令式语言
工具类别	一般编译工具
应用领域	通用、行业应用
仓库	https://github.com/tu-lang/tu

3.18.1 简介

凸语言是一款专为通用场景设计的动态编译型编程语言，项目于 2018 年启动，于 2022 年开源，目前已成功实现自举，处于试用优化阶段。

该语言的设计初衷源于对现有编程语言在追求高性能和安全性方面过度极端化的认识，例如 Rust 和 C++等。尽管这些语言在性能和安全性方面表现卓越，但在实际程序开发中给开发者带来了相当的负担，使人感到有些疲倦(凸|秃)。

相较之下，当前的动态语言如 PHP、Python 和 JavaScript 等虽然具有较高的开发效率，但性能通常较差，且以解释型为主，扩展性也受到一定限制。为了解决这些问题，不得不通过编写 C 语言的扩展库来实现一些底层特性。

凸语言的发展目标是在开发效率、性能和至简方面取得平衡。在开发效率方面，主要采用动态语法，无需繁琐的类型标注，使开发者能够专注于业务逻辑的实现。而在性能方面，则通过静态语法编写高性能库，为高性能场景提供了可行的解决方案，且提供了主流丰富安全的特性如栈式协程、多线程 GC。至简性是凸语言的另一个追求，它具有 100%零依赖，自给自足的特点，全链路实现自举（编译、汇编、链接），无需依赖外部工具链支持，目前可在任意 amd64 Linux 架构下灵活使用。

动态语法示例：

```
use fmt
use os
fn main(){
    map = {
      "1" : 'a',
      3   : 5 ,
      "hello" : "world" ,
      "arr"   : [ 0,1,2,3,4]
    }
    for k ,v : map {
        if k == "arr" {
            for v2 : v {}
        }
        fmt.println(k,v)
    }
    match map["hello"] {
        map     : os.die("not this one!")
        999     : os.die("not this one!")
        "hello" | "world": {
            fmt.println("got it",map["hello"])
        }
        _       : {
            os.die("not default")
        }
    }
}
```

```
$ tu build hello.tu

[ 10%] Compiling hello.tu v0.0.0
[ 30%] Compiler generate all Passed
[ 30%] Collecting object info
[ 40%] Checking symbol valid
[ 50%] Allocing address
[ 60%] Relocating symbol
[ 80%] Relocating address
[ 90%] Building executable binary
[ 100%] Generating executable binary
[ 100%] Finished hello.tu target(hello)

$ ./hello
3   5
hello   world
1   a
arr [0,1,2,3,4,]
got it  world
```

图 3.18-1

静态语法示例：

```
use std.atomic
mem Mutex {
   u32 key
}
Mutex::lock(){
   v<u32> = atomic.xchg(&this.key,mutex_locked)
   if v == mutex_unlocked {
      return Null
   }
   wait<u32> = v
   spin<u32> = active_spin

   loop {
      for i<i32> = 0 ; i < spin ; i += 1 {
         while this.key == mutex_unlocked {
               if atomic.cas(&this.key,mutex_unlocked,wait) != Null
                  return Null
         }
         procyield(active_spin_cnt)
      }
   }
   v = atomic.xchg(&this.key,mutex_sleeping)
   if v == mutex_unlocked return Null
   wait = mutex_sleeping
   futexsleep(&this.key,mutex_seeping,-1.(i8))
}
```

图 3.18-2

特性语法实例：

```
use runtime
use net.http.server

async handle(req , rsp) {
    header = req.GetHeader()
    stream = req.GetBodyStream()
    loop {
        buf = stream.read().await
        if buf == false break
        fmt.println(buf)
    }
    rsp.SendResp("hello world").await
}

fn dispatch(req , rsp){
    return handle(req , rsp)
}

fn main(){
    http = new server.Server("127.0.0.1" , 80)
    http.dispatch(dispatch)

    rt = runtime.new()
    rt.block(http.start())
}
```

图 3.18-3

接下来的时间，开发组将专注于以下方向：

- 完善多线程 Future 协程管理框架；
- 优化 Runtime 的多线程 GC 性能和稳定性；
- 丰富包依赖管理工具、文档手册；
- 企业实战应用项目；
- 设计好用的业务框架。

凸语言欢迎编程语言爱好者一起建设、打磨，共同发展一款属于我们开发人员自己的语言。

3.19 凹语言

项目分类	语言类、免费、开源（AGPL-3.0）、通用、接受社区贡献
语言类别	一般编程语言、命令式语言
工具类别	一般编译工具
应用领域	通用、计算机图形、建模与模拟、行业应用
主页	https://wa-lang.org/
仓库	https://gitcode.com/wa-lang/wa 、 https://github.com/wa-lang/wa

3.19.1 简介

凹语言（凹读音“Wā”），是针对 WebAssembly（简称 Wasm）设计的编程语言。从读音上看，“凹”是 Wasm 的首音节；从形状上看，“凹”字形似 Wasm 图标（方块上部缺个口）；凹语言起名的灵感，正是这种象形文字特有的、形状读音双重相似的巧合。

凹语言是由武汉凹语言科技有限公司（武汉市工科院科技园孵化器在孵企业）维护的开源项目，目标是：为高性能网页应用提供一门简洁、可靠、易用、强类型的编译型通用编程语言，目前处于工程试用阶段，项目重要里程碑如下：

- 2019 年，立项；
- 2022 年 7 月，正式开源；
- 2023 年 8 月，发布最小可用（MVP）版；
- 2024 年 11 月实现所有语法特性。

易用性是凹语言的设计重点——使用自动内存管理、字符串为基本类型等，均体现了这点。凹编译器是单文件可执行程序，内置工程脚手架，三步即可创建一个 Wasm 程序；此外凹语言提供了在线 Playground：https://wa-lang.org/playground ，在网页内即可编写、编译、运行、测试凹代码：

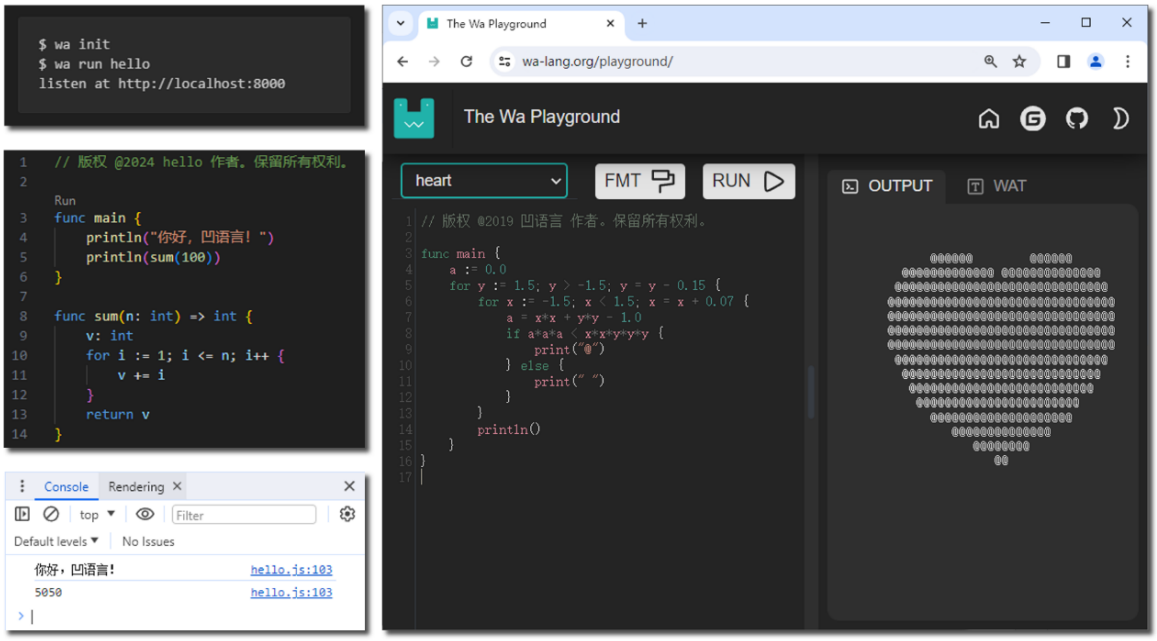


图 3.19-1

虽然处于早期阶段，凹语言仍展现了强劲的性能，使用它开发的任天堂 FC 模拟器：https://wa-lang.org/nes 可以流畅的运行各种 FC-ROM（作为对比，采用同样的模拟方法，Python 开发的 FC 模拟器性能仅有实机的 1%）：

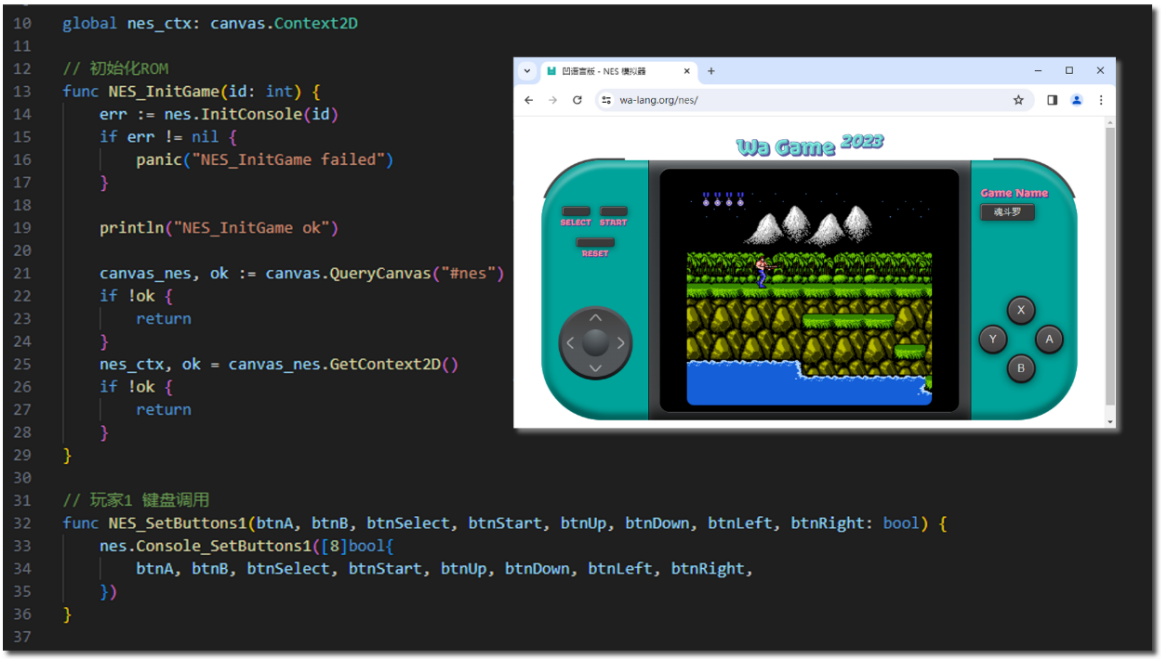


图 3.19-2

凹语言的应用方向包括 XR、游戏、工业设计、空间地信等需要高密度运算的网页应用，项目组正在为这些应用开发图形图像支持库：

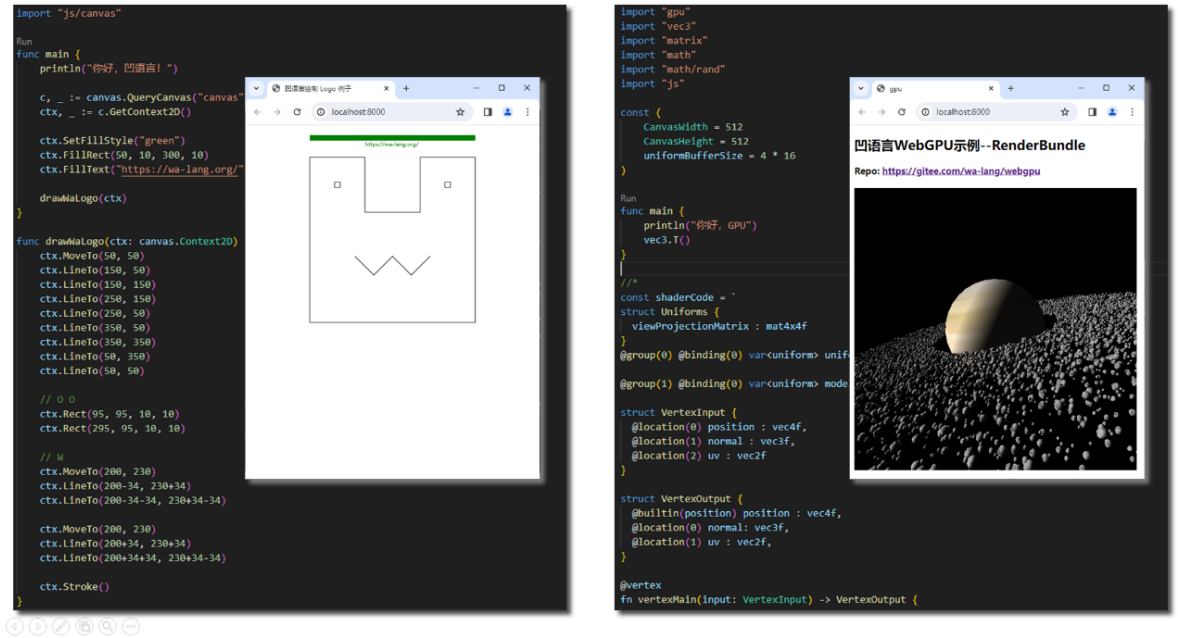


图 3.19-3

2019 至 2023，既凹语言第一个五年计划，开发组基本完成了“可用”的目标；对第二个五年计划，我们的目标是“好用”。作为第二个五年计划的开局之年，2024 年度项目进展主要体现在以下方面：

- 增加了函数重载、运算符重载、embed、map、defer、复数支持，实现所有语法特性；
- 开发了内置 wat 转 wasm 模块，实现编译器后端全自研；
- 提供了 p5、wasm4、Arduino Nano 33 等框架的实验性支持；
- 支持 GitHub Action 自动化构建。

接下来的时间，开发组将重点着力于以下方向：

- 功能性基础库、框架开发；
- 后端和运行时重构，性能提升；
- 实用项目落地；
- 工具链、手册、脚手架优化。

凹语言是社区合作开发的成果，编译器使用 Go 编写、标准库使用凹语言编写，随时欢迎编程语言爱好者围观、共建！

3.20 XLang

项目分类	语言类、免费、开源（AGPL-3.0）、通用、接受社区贡献
语言类别	通用编程语言（脚本语言）、命令式语言、函数式语言、可扩展语言
工具类别	解释器、代码生成
应用领域	通用、行业应用
主页	https://nop-platform.github.io/projects/nop-entropy/docs/dev-guide/xlang/
仓库	https://github.com/entropy-cloud/nop-entropy

3.20.1 简介

XLang 是一款混合采用 XML 标签语法和 JavaScript 语法的脚本语言，设计者为 canonical。XLang 语言是 Nop 低代码平台的底层支撑技术之一，它是世界上第一个内置了可逆计算理论差量合并算子支持的程序语言。XLang 版的 hello world 程序代码如下：

```
<c:log info="hello world" />
```

Nop 平台采用面向语言编程范式（Language Oriented Programming），在应用开发时并不是直接使用通用程序语言（如 Java、C#）来开发，而是先定义一个领域特定语言（DSL），然后再应用 DSL 来表达业务。为了快速开发和扩展 DSL 语言，我们需要一种能够定义 DSL 语法结构的元模型语言，同时还需要一系列的机制快速实现 DSL 的解释器、语法制导翻译等。XLang 语言包含了 XDef 元模型定义语言，Xpl 模板语言，XScript 表达式语言和 XTransform 结构转换语言等子语言，它们共同构成一个完整的 DSL 开发基础设施。通过增加简单的 XDef 元模型定义，我们就可以自动得到对应的 DSL 解析器、验证器、IDE 插件、调试工具等，并自动为 DSL 领域语言增加模块分解、差量定制、元编程等通用语言特性。

- 通过 XDef 元模型定义新的 DSL：

```
<!--
状态机模型 DSL
@initial 初始状态的 id
@stateProp 实体上的状态属性名
-->
<state-machine initial="!var-name" stateProp="!string"
ignoreUnknownTransition="!boolean=false"
               xdef:name="StateMachineModel" xdef:bean-
package="io.nop.fsm.model"
               x:schema="/nop/schema/xdef.xdef"
xmlns:x="/nop/schema/xdsl.xdef" xmlns:xdef="/nop/schema/xdef.xdef"
>
    ...
    <state id="!var-name" xdef:unique-attr="id" xdef:ref="StateModel"/>

    <!-- 进入状态时触发的监听函数 -->
    <on-entry xdef:value="xpl"/>

    <!-- 离开状态时触发的监听函数 -->
    <on-exit xdef:value="xpl"/>

    <!--
    状态迁移出现异常时触发的监听函数。如果返回 true，则认为异常已经被处理，不对外抛
出异常
    -->
    <handle-error xdef:value="xpl-fn:(err)=>boolean"/>
</state-machine>
```

- XML 和表达式语法的相互嵌入。XLang 没有采用 jsx 语法实现类 XML 语法，而是沿用 XML 语法，扩展 JavaScript 中的 Template 表达式语法：

```
let resut = xpl `<my:MyTag a='1' />`
const y = result + 3;
```

等价于

```
<my:MyTag a='1' xpl:return="result" />
<c:script>
  const y = result + 3;
</c:script>
```

XLang 的整体结构严格符合 XML 格式要求，因此它作为模板语言来生成 XML 时满足 Lisp 语言所首创的同像性（Homoiconicity），因此特别适合元编程和宏函数的实现。

XLang 修改了 JavaScript 中的 Template 表达式语法的解析格式，将反引号字符之间的内容识别为一个在编译期待解析的字符串，而不是一个 Expression 列表。这使得 XLang 可以利用这个语法形式扩展支持更多的 DSL 格式，比如引入类似 C#的 LinQ 语法

```
const result = linq `select sum(amount) from myList where status >
${status}`
```

实现这种类似 linq 语法的解析器非常简单，只需要在 Java 中定义一个静态函数，标记为 `@Macro`

```
    @Description("编译并执行 xpl 语言片段，outputMode=none")
    @Macro
    public static Expression xpl(@Name("scope") IXLangCompileScope scope,
        @Name("expr") CallExpression expr) {
        return TemplateMacroImpls.xpl(scope, expr);
    }
```

- 宏函数和编译期执行。XLang 支持编译期元编程，允许在编译期执行图灵完备的代码，并动态生成新的待编译的语法结构。

```
 <!--在编译期解析标签体得到 ValidatorModel，保存为编译期的变量 validatorModel-->
<c:script><![CDATA[
import io.nop.biz.lib.BizValidatorHelper;

let validatorModel = BizValidatorHelper.parseValidator(slot_default);
// 得到<c:script>对应的抽象语法树
let ast = xpl `
   <c:ast>
    <c:script>
      import io.nop.biz.lib.BizValidatorHelper;
      if(obj == '$scope') obj = $scope;
      BizValidatorHelper.runValidatorModel(validatorModel,obj,svcCtx);
    </c:script>
   </c:ast>
`
// 将抽象语法树中的标识名称替换为编译期解析得到的模型对象。这样在运行期就不需要动态加载模型并解析
return ast.replaceIdentifier("validatorModel",validatorModel);
    ]]></c:script>
```

- XDSL 的差量生成与合并机制。Nop 平台中所有的 DSL 都支持 x-extends 差量合并机制，通过它实现了可逆计算理论所要求的计算模式：

```
> App = Delta x-extends Generator<DSL>
```

具体来说，所有的 DSL 都支持 `x:gen-extends` 和 `x:post-extends`

配置段，它们是编译期执行的 Generator，利用 XPL 模板语言来动态生成模型节点，允许一次性生成多个节点，然后依次进行合并，具体合并顺序定义如下：

```
<model x:extends="A,B">
    <x:gen-extends>
        <C/>
        <D/>
    </x:gen-extends>

    <x:post-extends>
        <E/>
        <F/>
    </x:post-extends>
</model>
```

合并结果为：

```
F x-extends E x-extends model x-extends D
              x-extends C x-extends B x-extends A
```

当前模型会覆盖 `x:gen-extends` 和 `x:extends` 的结果，而 `x:post-extends` 会覆盖当前模型。

借助于 `x:extends` 和 `x:gen-extends` ，我们可以有效的实现 DSL 的分解和组合。具体介绍参见 ： https://zhuanlan.zhihu.com/p/612512300。

- 可扩展语法。类似于 Lisp 语言，可以通过宏函数和标签函数等机制扩展 XLang 的语法。可以通过 `<c:lib>` 来引入新的语法节点，然后在该节点内部再通过宏函数等机制实现结构转换。

```
<c:lib from="/nop/core/xlib/biz.xlib" />
<biz:Validator fatalSeverity="100"
               obj="${entity}">
    <check id="checkTransferCode" errorCode="test.not-transfer-code"
           errorDescription="扫入的码不是流转码">
        <eq name="entity.flowMode" value="1"/>
    </check>
</biz:Validator>
```

`<biz:Validator>` 引入一个验证用的 DSL，Validator 标签在编译的时候会利用宏函数机制解析节点内容，将它翻译为 XLang 的 Expression 来运行。

第四章　关于我们

PLOC 介绍：

编程语言开放社区（Programming Language Open Community，简称 PLOC）是由国内从业者于 2023 年底自发组建的、编程语言及编译器专业社区，社区组建基于这样一些现实：

- 编程语言和编译器是软件行业真正的“根技术”和“工业母机”，但我国在该行业几无建树；
- 决策层和工业界开始意识到 PL 技术的重要性；
- 现今被广泛使用的编程语言，发展经验无规律可循；

目前国内存在很多编程语言项目，但从业者高度分散，无人能给出有代表性的、全面的全景图，也缺乏行业发声渠道。

与其他松散兴趣组不同，PLOC 社区理事会以项目为组成成员，有明确的纲领和愿景、完整的制度章程、严密的组织结构，常设有决策机构（社区理事会）、综合服务机构（理事会秘书处），通过专业委员会推动社区建设和活动开展。

“从业者互助”是 PLOC 的核心理念。编程语言项目启动门槛高、商业化难度高，从业人数相对稀少，因此业内的交流和互相支持尤为重要，我们希望藉由本社区的成立，能推动更多的编程语言项目启动（增大基数）、延长项目存活时间（提高成功系数），助力国产根软件发展。

湖北省软件行业协会介绍：

湖北省软件行业协会（HBSIA）成立于 2000 年，是一家从事软件研发及信息服务的 5A 级社会组织。

成员组成：协会现有正式会员单位约 1200 家，会员单位均为从事软件及信息服务研发、销售企业、科研等相关经济组织；现任理事长（法人代表）为烽火通信科技股份有限公司董事长曾军。

社会认可：2010 年被民政部授予“全国先进社会组织”称号，2017 年被共青团中央授予“全国青年文明号”称号，2012 年、2024 年两度被湖北省民政厅评定为“5A 级社会组织”；

服务品牌：多年来，湖北省软件行业协会以“争当社会组织服务旗帜、打造国家一流软件行业协会、服务软件产业高质量发展”为目标，坚持职业化规范化市场化，以“场景路计划”作为长远规划，搭建了由石榴会、专委会和技术供应链平台、市场服务平台、人才服务平台、资本服务平台、软件和信息化解决方案荟萃平台等构成的“2 会 10 平台”服务体系，服务数字经济相关领域的行政机关、事业单位、科研院所、高校、企业约 2500 余家，服务对象遍及全国二十多个省、直辖市、自治区。协会在加强党建引领、助力政府决策、促进行业自律、完善协商民主、培育数字人才、实施就业优先、研制数字标准、搭建交流合作、赋能企业发展、履行公益责任等方面发挥积极作用，被民政部领导称为“社会组织示范标杆”。

附 录

语言类别:

a. 通用编程语言　　b. 并行语言　　c. 并发语言

d. 分布式语言　　e. 命令式语言　　f. 面向对象语言

g. 函数式语言　　h. 约束和逻辑语言　　i. 数据流语言

j. 可扩展语言　　k. 汇编语言

工具类别:

a. 一般编译工具　　b. 解释器　　c. 增量编译器

d. 可重定向编译器　　e. 实时编译器　　f. 动态编译器

g. 生成器　　h. 代码生成　　i. 运行时环境

j. 预处理器　　k. 解析器

应用领域:

a. 通用　　b. 计算理论　　c. 计算数学

d. 网络　　e. 信息系统　　f. 安全

g. 机器学习　　h. 人工智能　　i. 并行计算

j. 并发计算　　k. 分布式计算　　l. 建模与模拟

m. 计算机图形　　n. 行业应用

本文电子版：
https://cdn-static.gitcode.com/doc/CNPL-2024-CHS.pdf

编程语言开放社区（PLOC）
微信公众号

湖北省软件行业协会
微信公众号