国产编程语言蓝皮书

2023

目录

版权声明

- 特别顾问：梁宇宁
- 策划：柴树杉、丁尔男、杨海龙
- 编辑：陈朝臣、杜天微、贺师俊、李登淳、刘小东、刘永康、题叶、吴森、徐鹏飞、杨海玲、赵普明、朱子润
- 支持单位：Gitee（https://gitee.com/）
- 赞助单位：武汉市江夏区凹语言开发工作室

第一章 前言

1.1 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，编程语言开放社区（PLOC）编写了《国产编程语言蓝皮书》-2023（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。

“从业者自治”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。

为保持信息时效性性，蓝皮书将持续更新发布。本文是 PLOC 社区编写国产编程语言黄页的首次尝试，编委组织、文档结构、内容模板等均存在不足之处，希望各界不吝赐教，使蓝皮书能逐步完善，更好的帮助国产编程语言项目提高存活几率，助力国产根软件的发展。

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. 项目为编程语言或编译工具；
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。

蓝皮书工作区仓库地址：https://gitee.com/ploc-org/CNPL-2023

> 《国产编程语言蓝皮书》-2023 编委会对收录标准拥有最终解释权。

1.4 收录方法

满足收录标准的项目可在以下地址中增加项目同名目录： https://gitee.com/ploc-org/CNPL-2023/tree/master/projects，将项目简介等资料以 markdown 格式填入其中，填写要求及案例见：https://gitee.com/ploc-org/CNPL-2023/tree/master/projects/sample。

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、联系方式可用；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

- 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：附录一 语言类别）
- 工具类别（详细清单见：附录二 工具类别）
- 应用领域（详细清单见：附录三 应用领域）

第二章 项目列表

蓝皮书本次共收录项目 15 个，各项目名称如下（按字母排序）：

- Aya
- Calcit
- CovScript 智锐
- Deeplang
- HVML
- K 语言
- KCL
- NASL
- Z 语言
- 凹语言
- 洛书
- 青语言
- 狮偶
- 凸语言
- 豫言

2.1 Aya

项目分类	免费、开源（MIT）、通用、接受社区贡献
语言类别	高级编程语言、函数式语言、多范式语言
工具类别	类型检查器、解释器
应用领域	通用、行业应用、计算数学
主页	https://www.aya-prover.org/
仓库	https://github.com/aya-prover/aya-dev

2.1.1 简介

Aya 语言的特色是它强大的类型系统。这个类型系统拥有『依值类型（dependent type）』、『参数化多态（polymorphism）』这两个功能，

且支持等号类型。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，且它等价于函数 `(x : 列表) -> 插入排序(x) = 归并排序(x)`，

也就是接收一个列表、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。

在 https://github.com/lazyparser/weloveinterns/blob/master/bunbun.md（Aya 团队的招募贴）中，有更详细的项目动机介绍。

有关 Aya 中语言特性的学术论文参见 https://www.aya-prover.org/pubs 这个页面。

2.2 Calcit

项目分类	免费、开源（MIT）、通用、接受社区贡献
语言类别	一般编程语言、脚本语言
工具类别	JavaScript 生成工具，解释器
应用领域	Web 开发
主页	https://calcit-lang.org/
仓库	https://github.com/calcit-lang/calcit

2.2.1 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：

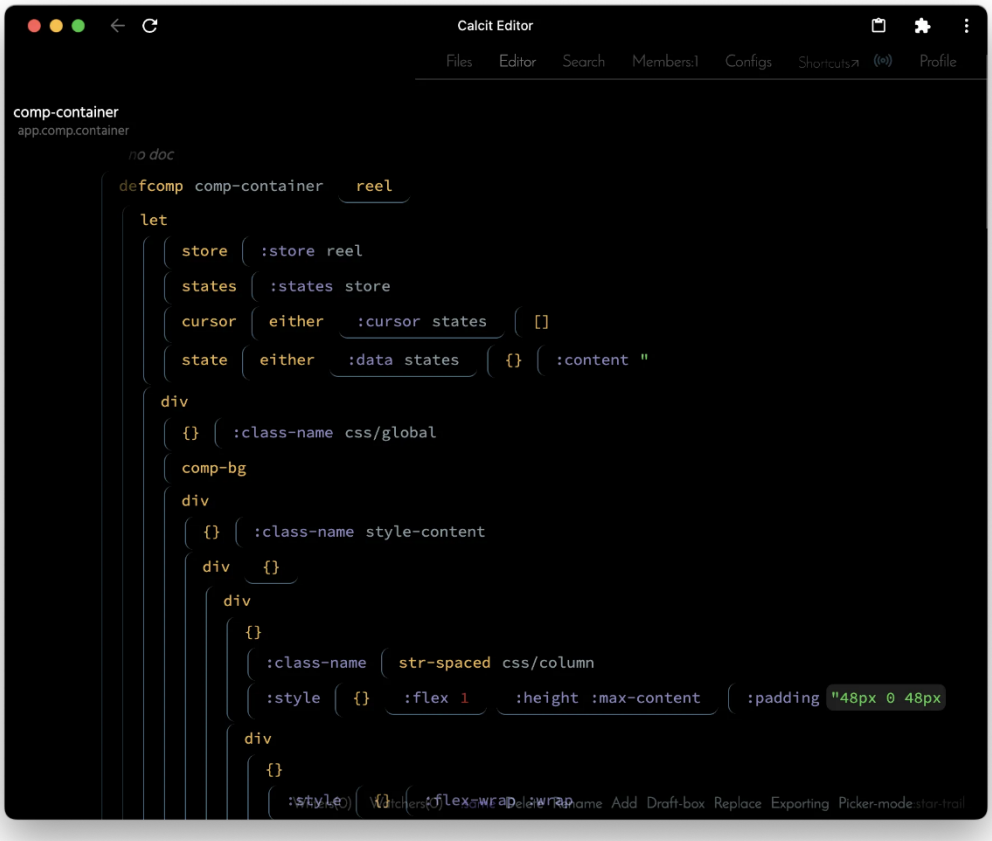

图 2.2-1

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。

2.3 CovScript 智锐

项目分类	免费、开源（Apache 2.0）、通用、接受社区贡献
语言类别	命令式语言、多范式语言
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

2.3.1 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
```

图 2.3-1

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。

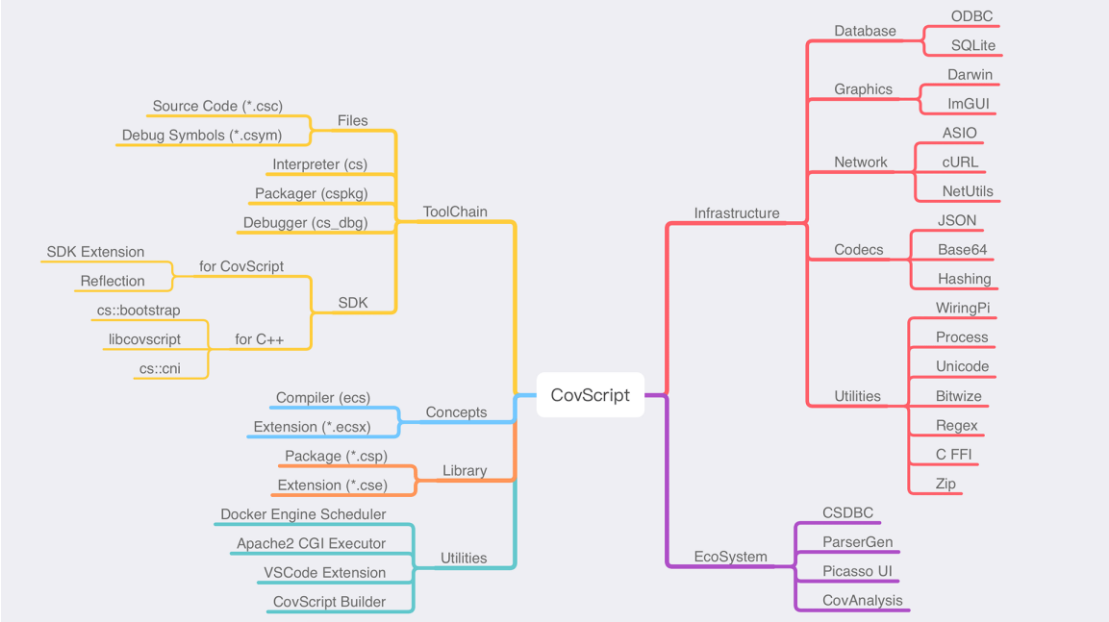

图 2.3-2

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。

2.4 Deeplang

项目分类	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	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

2.4.1 简介

Deeplang 语言是具有鲜明内存安全特性的面向 IoT 场景的语言，设计过程中参考 Rust 的安全机制，但又根据 IoT 场景的特性选择了更合适的编译执行模式。Deeplang 是一种静态类型、强类型语言，参考 C-style 设计语法，同时支持过程式、逻辑式和函数式的混合范式。

Deeplang 编译和运行的软件分别为 Deepc 和 DeepVM。由于 Deeplang 的目的是探索资源受限场景下语言特性的设计，现阶段的 Deepc 和 DeepVM 只能用于语言特性研究，并不具备任何商用能力。

Deeplang 的编译。Deepc 是由 Ocaml 开发的。首先 Deeplang 源码经过语法解析器（parser）生成语法树，遍历器（walker）对语法树进行多次遍历获取相关的符号表，转换模块（conversion）基于符号表将语法树转译成 ANF IR，最后 codegen 将 ANF IR 转化成 WASM 字节码。Deepc 暂时只有类型检查器（type checker），没有类型推断器（type infer），无法推断类型信息，因此所有 Deeplang 源码需要主动标注类型信息，否则视为语法错误。

Deeplang 的运行。DeepVM 是由 C 语言开发的，支持 WASM1.0，包括字节码加载、内存管理、解释执行、FFI 机制。

ADT 语法示例：

```
// 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 } };
}
```

图 2.4-1

Interface 语法示例：

```
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
}
```

图 2.4-2

Pattern Match 语法示例：

```
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; }
  }
}
```

图 2.4-3

2.5 HVML

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

2.5.1 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 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>

        <!-- the `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>
```

图 2.5-1

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：

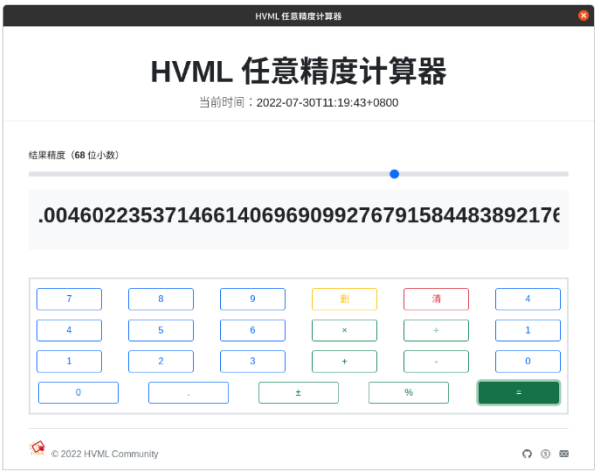

图 2.5-2

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

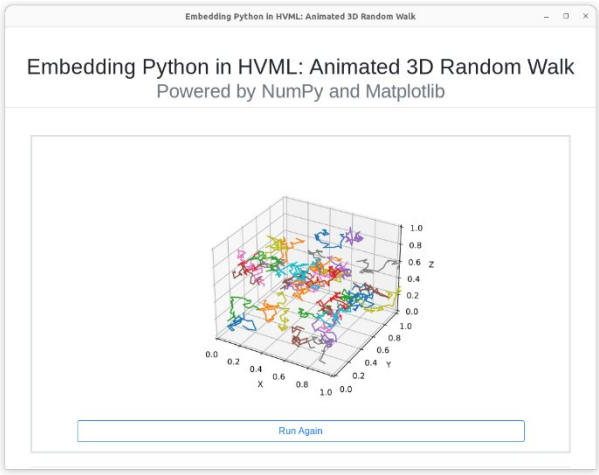

图 2.5-3

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。

2.6 K 语言

项目分类	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	一般编译工具
应用领域	通用
仓库	https://github.com/kulics-works/k

2.6.1 简介

K 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 K 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 K 语言尝试了几个比较有意思的设计：
- 前置括号的泛型语法
- 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 (T1, T2)Pair(left: T1, right: T2);

let main() = {
    lef a1: (Int, Int)Pair = (Int, Int)Pair(1, 2);
    ## a1.left: Int, a1.right: Int
    lef a2: (Bool, Bool)Pair = (Bool, Bool)Pair(true, false);
    ## a2.left: Bool, a2.right: Bool
    lef a3: (Int, String)Pair = (Int, String)Pair(1, "a");
    ## a3.left: Int, a3.right: String
```

```
}
```

代码示例 2：

```
let main() = {
    if true || f() do {
        ...
    }
    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
}
```

2.7 KCL

项目分类	免费、开源（Apache-2.0）、专用、接受社区贡献
语言类别	一般编程语言、声明式语言
工具类别	一般编译工具
应用领域	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

2.7.1 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

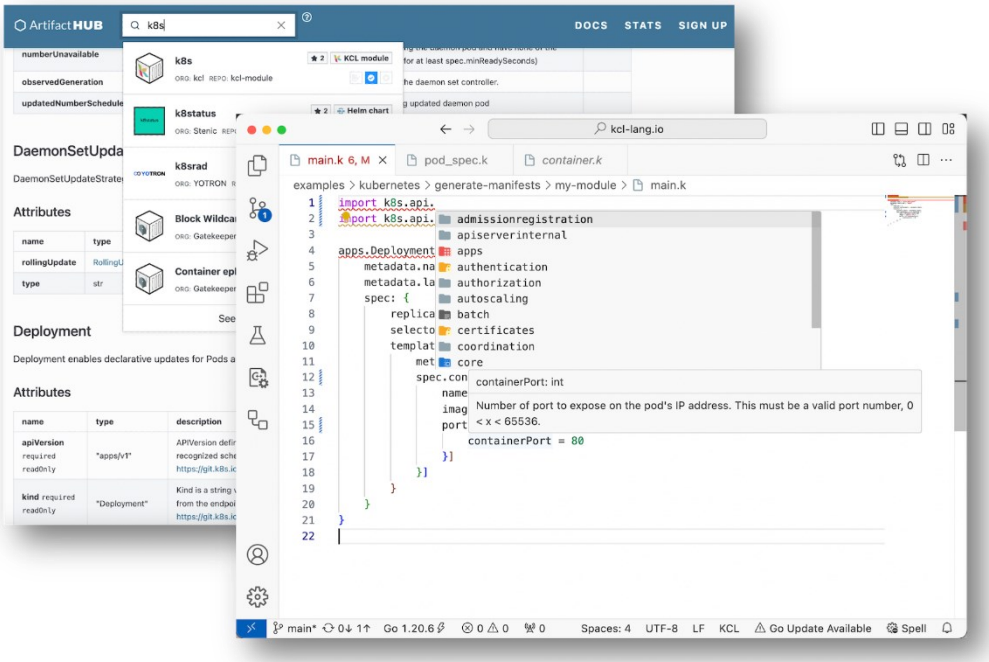
Artifact HUB
k8s
DOCS
STATS
SIGN UP
k8s
k8status
k8srad
Block Wildca
Container epl
DaemonSetUpda
Attributes
Deployment
Attributes
kcl-lang.io
main.k 6, M
pod_spec.k
container.k
examples > kubernetes > generate-manifests > my-module > main.k
import k8s.api.
import k8s.api.
apps.Deployment
metadata.na
metadata.la
spec: {
replica
selecto
templat
met
spec.con
name
imag
port
containerPort = 80
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.
main*
Go 1.20.6
Spaces: 4
UTF-8
LF
KCL
Go Update Available
Spell

图 2.7-1

2.8 NASL

项目分类	共享(免费试用无时长限制)、闭源、专用、不接受社区贡献
语言类别	多范式语言、可扩展语言、高级编程语言
工具类别	代码生成、增量编译器
应用领域	行业应用
主页	https://nasl.codewave.163.com/

2.8.1 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：

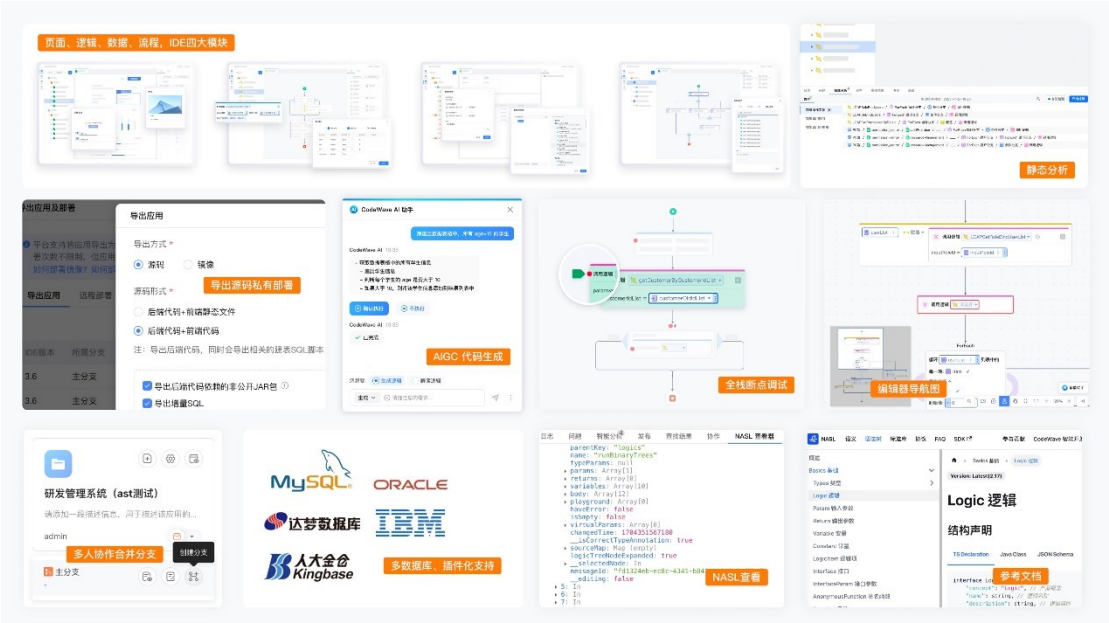

图 2.8-1

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：

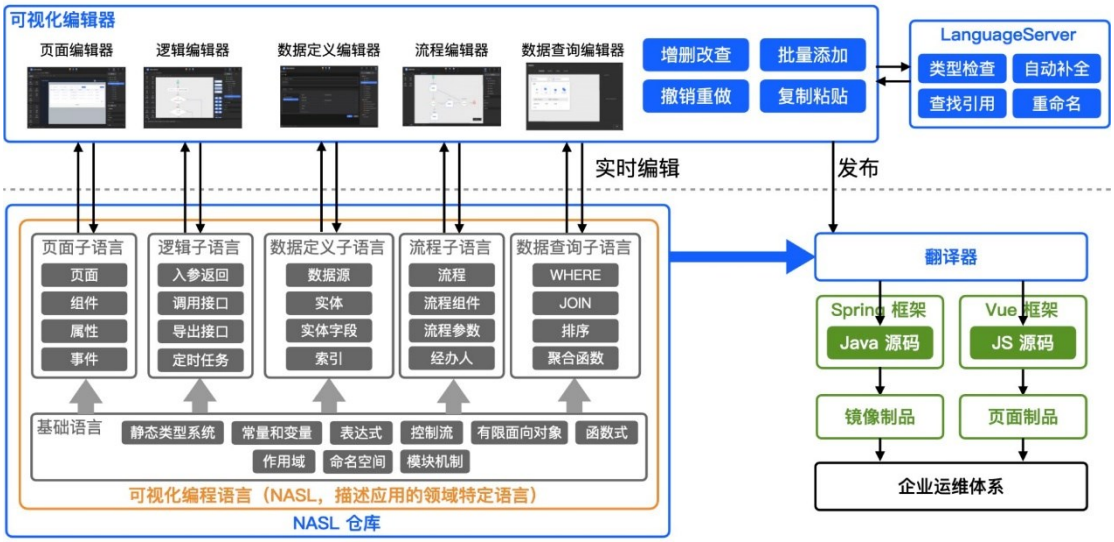

图 2.8-2

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。

2.9 Z 语言

项目分类	免费、开源（MIT）、专用、接受社区贡献
语言类别	一般编程语言
工具类别	一般编译工具、解释器、转译器
应用领域	图形引擎、AI、教学
仓库	https://gitee.com/z-lang/zc

2.9.1 简介

到目前为止，Z 语言还是一门玩具语言，但它的野心很大。Z 语言的目标，是做成“最强的玩具语言”。

何谓玩具语言？与通用编程语言相比，Z 语言有如下特征：

- 定位：玩具语言是用来玩的，是用来分享编程语言相关的知识的。玩具语言可以用来作为教学案例，但不用来编写大规模工业应用。
- 可读性：基于前面的定位，玩具语言需要抛弃掉所有影响可读性的特性，包括性能优化、安全相关的问题，而只关注语言特性本身。
- 教学性：我给 Z 语言配套了一本同步的开源书[《Z 语言炼成记》](https://gitee.com/z-lang/devlog)。Z 语言编译器本身遵循“增量开发”原则，同步书也增量更新。

那么何谓“最强”呢？这里的“最强”，指的是和其他玩具语言相比，Z 语言有如下特点：

- 麻雀虽小，五脏俱全。
  - 语言特性上，Z 语言支持动态类型和静态类型，并支持常见的函数、类、接口、泛型、模块化等语言特性。

  - 工具链上，Z 语言既可以解释执行，也可以直接编译成汇编（当前支持 Windows 和 Linux 的 X86_64 汇编），还可以转译成 C、Python 和 JavaScript。

- 应用：玩具语言也可以有应用，即“玩具应用”。Z 语言的驱动应用是一个 AI 图形引擎[Zaige](https://gitee.com/z-lang/zaige)。当然，这个引擎本身的定位也是“玩具引擎”，并且也有自己的同步开源书[从零开始制作 AI 图形引擎](https://gitee.com/z-lang/zaige_book)。

- 创新的语言特性：

  - 面向场景编程。根据不同的场景，编译器提供不同的语言特性和相关的库。例如，编写 AI 绘图脚本时，编译器会自动将 Z 脚本转换为 Python 脚本，然后调用 Python 的 AI 绘图库。

  - 编译期脚本执行。在编译或解释过程中，编译器可以调用 Z 解释器，执行任意 Z 函数。这个特性可以用来实现泛型、模版元编程，以及其他类型的代码生成功能。

  - 生态融合：生态融合：Z 语言既可以解释执行，又可以转译成 C、Python 和 JavaScript，因此 Z 语言可以作为一门胶水语言，方便地融合多个生态。

以上内容大部分都是构思（吹牛），还没有实现。Z 语言从 2023 年 11 月正式立项，到目前为止，进度如下：

- 语法特性实现了最基本的部分：基本类型、变量、函数、类。可以算是一门最简单的玩具语言了。从代码推送可以看到，现在实现了 34 个小版本，最新版是 v0.0.34。

- 工具链：编译器、解释器和转译器都同步实现了。

- 同步开源书：《Z 语言炼成记》更新了 5 章，和编译器的进度保持一致。

- Zaige 引擎：完成了初始的调研和学习工作，正式开始编码。初步目标是实现一个简单的 2D 图形引擎，以及简单的瓷砖地块编辑器。

- 同步开源书：《从零开始制作 AI 图形引擎》完成了前三章的编写，内容是总体设计和相关领域的调研。

Z 语言的示例代码如下：

```
use io.[print, open, W] // 导入标准库，用 use 关键字。
const PI = 3.1415926f // 常量定义用 const 关键字。
```

```
// 函数定义和 Go 比较像。函数的关键字是`fn`。Z 语言里函数默认是纯函数，即不能产生副作用。
fn add(a int, b int) int {
  a + b // 代码块的最后一个语句即是返回值
}

@mut // 加上@mut 标注，则函数可以产生副作用
fn writeFile(name str, s str) bool {
  let f = open(name, W)
  on(exit) { f.close(); return false }
  f.write(s)
  true
}

@var // 加上@var 标注，则函数可以不指定参数和返回值类型，这样的函数和 JS 与 Python 的函数类似。
fn alert(message) {
  message = "Alert!! $message"
  print(message)
}

fn main { // main 函数是特殊的函数，它是程序的入口。
  print("Hello, world!") // 语句结尾不需要';'。函数调用形式与 C 一致。
  print("Here is pi: ${PI}") // Z 语言支持嵌入字符串。这里常量 PI 的值被直接嵌入到字符串里了。
  let a int = 10 // let 标量，类似于 C 的变量，但它的值是不可变的。
  a = 12 // 错误！a 是不可变的量。
  mut b = 5 // mut 变量。这个相当于 C 语言里的普通变量。Z 支持基本的类型推导
  b = b * 2 // 正确！b 是可变量。
  b = "Z 语言" // 错误！不能改变变量的类型。
  var c = 5 // var 幻量。相当于 JS 里的 var 变量，不但值可以变，类型也可以变。
  c = "Z 语言" // 正确！c 是幻量，可以从整数类型变成字符串。
  print("a+b is $add(a, b)")
}
```

2.10 凹语言

项目分类	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、代码生成
应用领域	通用、建模与模拟、计算机图形
主页	https://wa-lang.org/
仓库	https://gitee.com/wa-lang/wa

2.10.1 简介

凹语言 是针对 WebAssembly 设计的、静态数据类型的、编译型通用编程语言，目标是：

- 简单易用
- 强表达力
- 高性能
- 可用于工业程序开发

项目起意于 2019 年；2022 年 1 月正式启动、7 月开源；2023 年 8 月发布最小可用（MVP）版；目前处于工程试用阶段。

凹语言的设计重点是易用性，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：

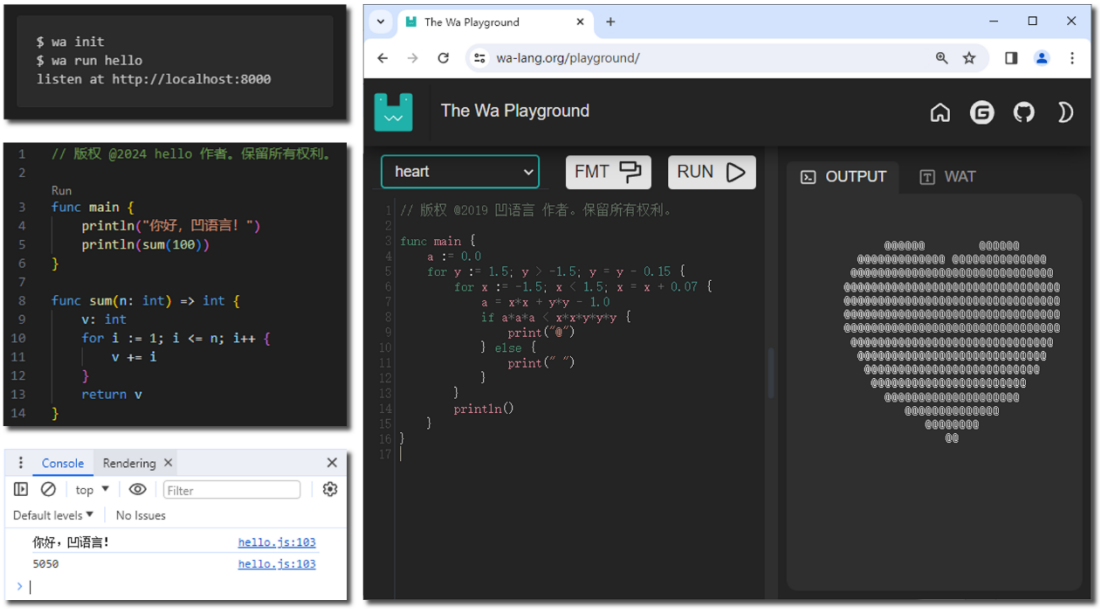

图 2.10-1

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%）：

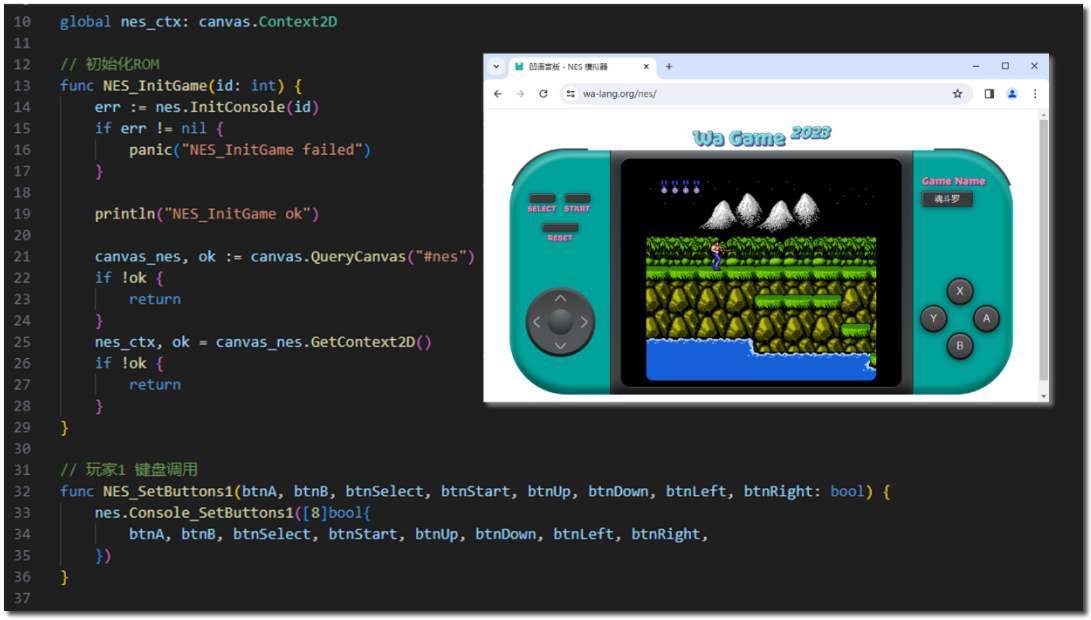

图 2.10-2

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等支持库：

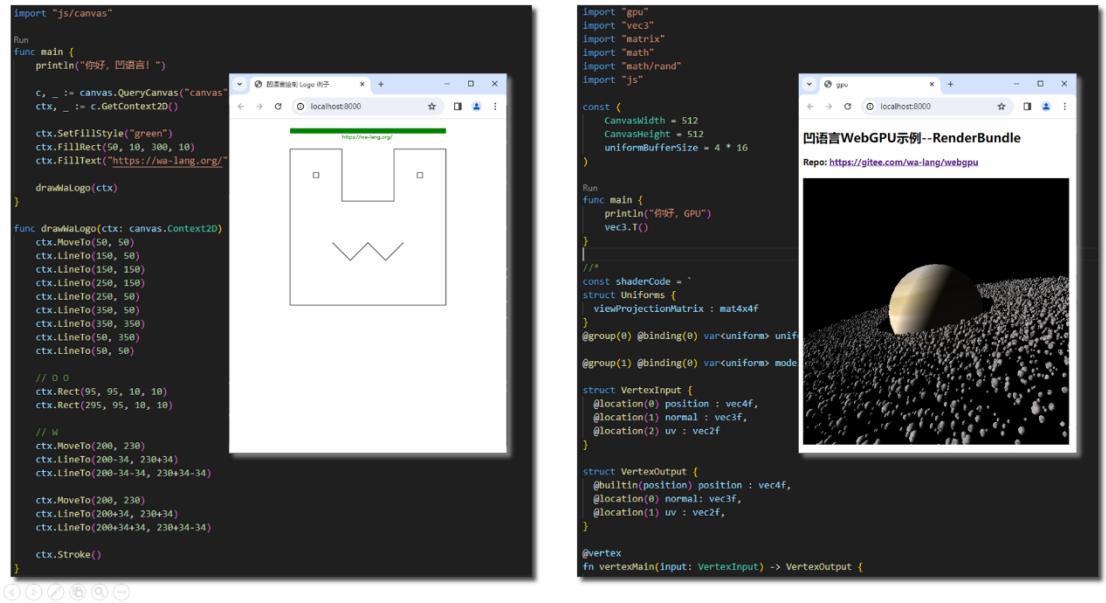

图 2.10-3

凹语言是社区合作开发的成果，编译器使用 Golang 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。

2.11 洛书

项目分类	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
主页	https://losu.tech/
仓库	https://gitee.com/chen-chaochen/lpk

2.11.1 简介

洛书编程语言，原名洛书汉语编程，英文缩写 Losu（Language of System Units），最早发布于 2020 年，项目目标是打造开源、高效、强大的国产编程语言。目前洛书已经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、在线 Playground 与软件源，初步成为一门具备实用性的超轻量级、跨平台脚本语言。

洛书的核心设计理念是易用与现代化，并兼顾中文程序设计，目前，洛书已经可以实现：

- 类 Python 关键词，动态数据类型与动态语法特性，图灵完备、支持面向过程、面向对象与部分元编程的特性。
- 支持常见语法结构，包括 变量声明、方法调用、对象、数组、字符串、表达式、控制语句
- 支持 运算符重载、构造函数、匿名函数、闭包、高阶函数、柯理化、鸭子类型等特性，并且使用简洁的方式进行实现
- 完善的 UTF-8 编码支持，支持使用 字母、中文、甚至 emoji 作为代码，在 Windows 平台提供可选的 GBK/UTF-8 转换功能
- 内置 GC，自动化的内存管理机制

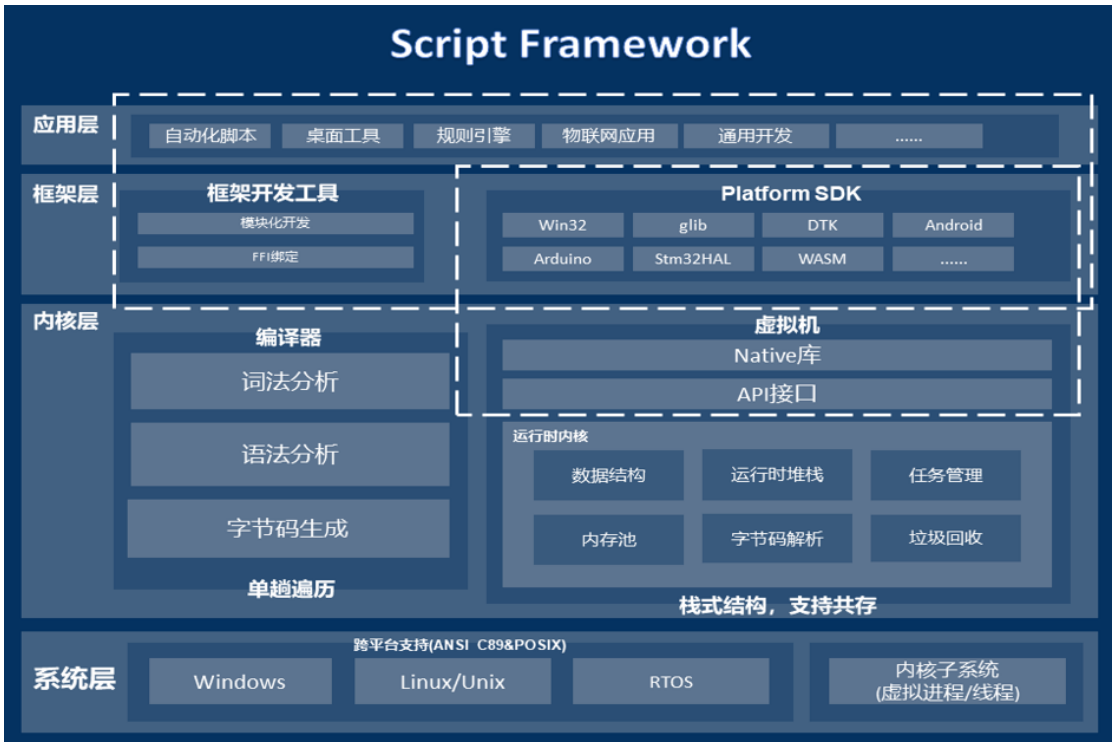

图 2.11-1

洛书是一门富有创新性的编程语言，拥有简洁、高效且可靠的实现。洛书拥有良好的单线程性能，支持轻量化的协程，和真正的多线程（洛书抛弃了传统脚本语言的 GIL 全局解释锁，并且为每个线程分配了独立的 GC）。

图 2.11-2

与此同时，洛书的零依赖、易移植与超低的运行开销特性，使得其不仅支持 PC 设备

Windows 与 Linux 等 OS，也可运行在多种低资源 MCU、RTOS 乃至裸机环境中，并且维持优秀的运行效率（高于 micropython）。

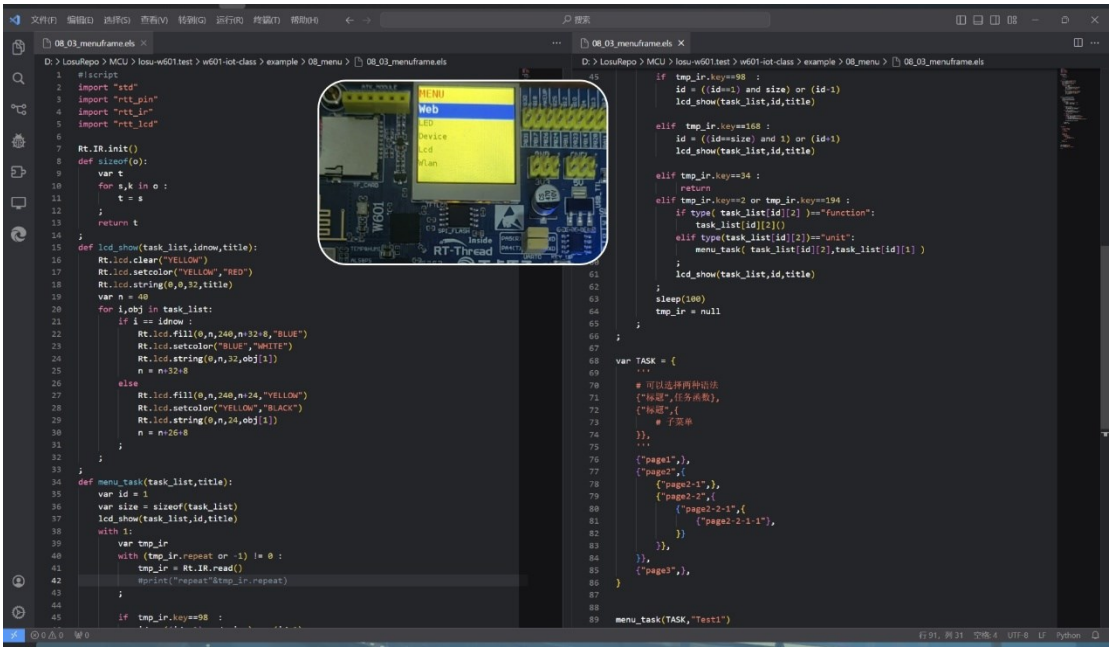

图 2.11-3

洛书已经打造出独立、开源的核心生态（脚本引擎、核心库、基础工具），且在中华人民共和国国家版权局注册，登记号：2023SR0953209。此外，洛书积极与国内高校创新团队进行合作，对洛书的国产软硬件生态的兼容性进行了大量改进优化工作，基于洛书制作的作品在“统信杯”计算机设计大赛信创赛道获得国家级奖项，并产出多篇软著、专利等成果。

目前，洛书可以支持 UOS、Deepin、Rt-thread 等多种国产 OS，龙芯、鲲鹏、玄铁等多种国产处理器，覆盖 Arm、x86、C-Sky 等多种指令集。

2.12 青语言

项目分类	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

2.12.1 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介，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：

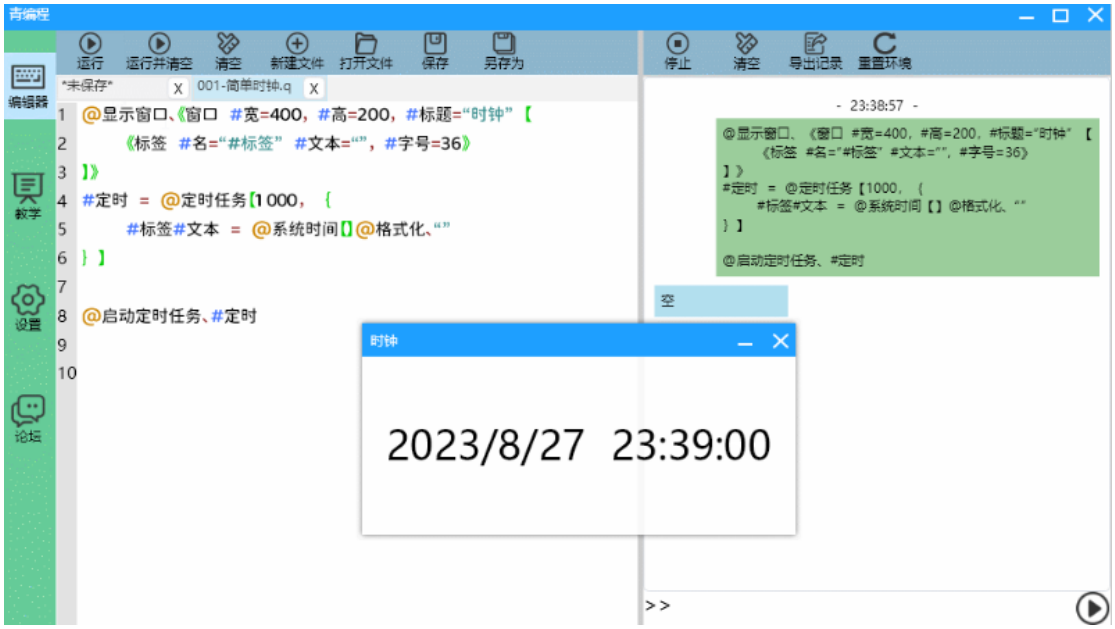

图 2.12-1

AI 图片分类：

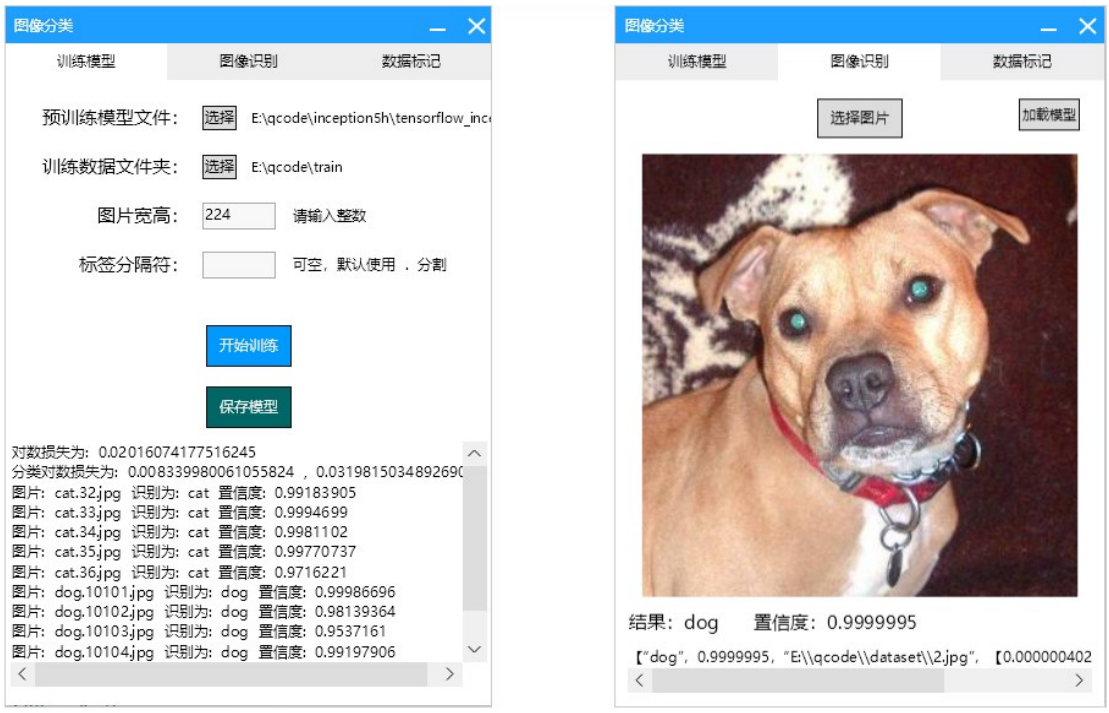

图 2.12-2

运行大语言模型：

图 2.12-3

2.13 狮偶

项目分类	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、高级编程语言
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

2.13.1 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。

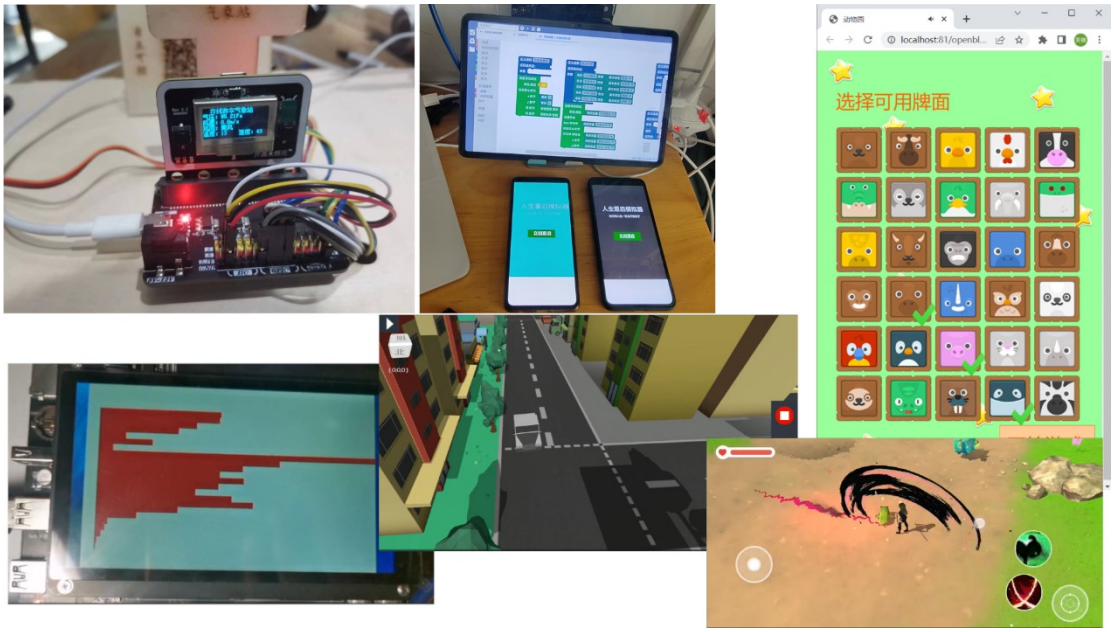

图 2.13-1

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。

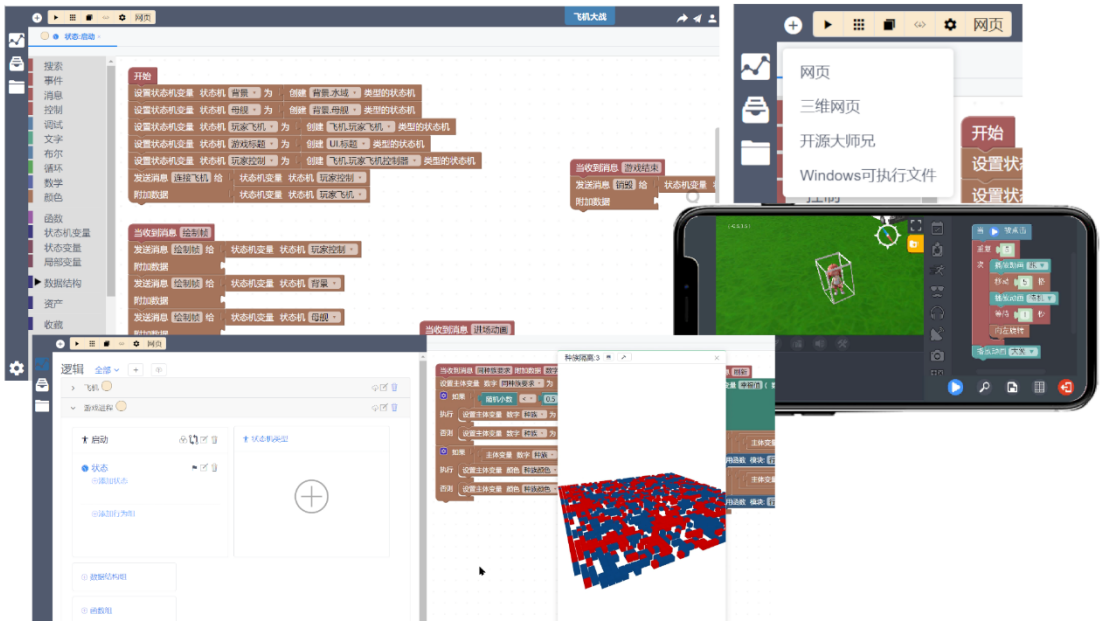

图 2.13-2

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。

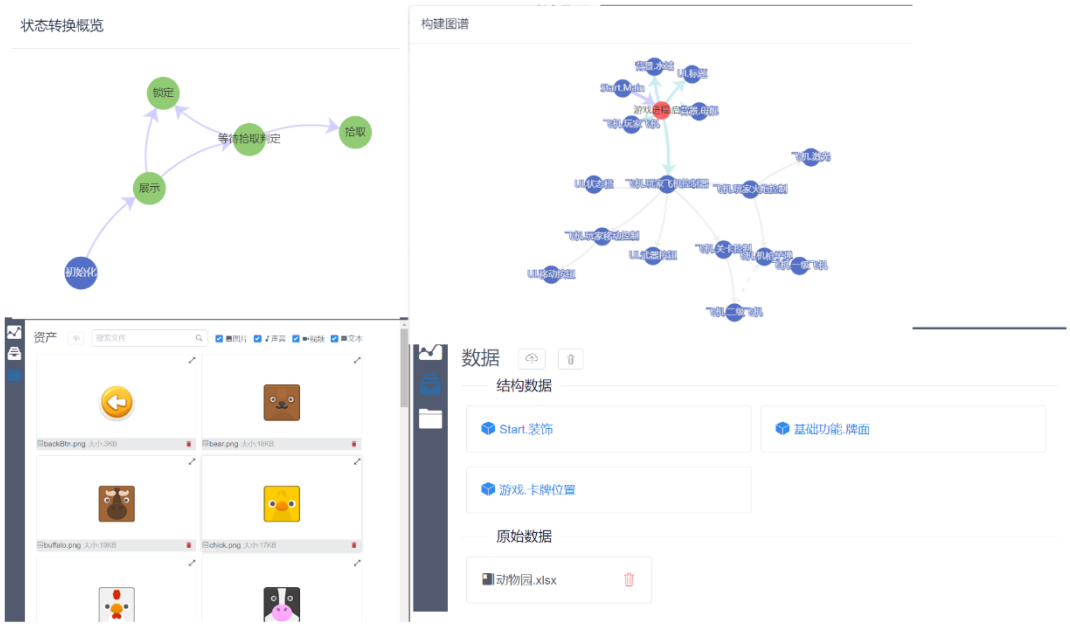

图 2.13-3

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。

2.14 凸语言

项目分类	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://gihub.com/tu-lang/tu

2.14.1 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。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 2.14-1

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 2.14-2

特性语法实例：

```
use asyncstd.runtime
use asyncstd.std
use asyncstd.http.server

async fn handle(req , rsp) {
   header = req.GetHeader()
   stream   = req.GetBodyStream()
   loop {
      data = stream.read()
      if data == false break
      std.sleepus(100)
      fmt.println(data)
   }
   rsp.SendResp("hello world")
}
fn main() {
   http = new server.Server("127.0.0.1" , 8080)
   http.request(handle())

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

async

图 2.14-3

2.15 豫言

项目分类	开源（AGPL）、通用
语言类别	一般编程语言、高级语言、函数式语言
工具类别	一般编译工具
应用领域	编程语言与中文编程研究
主页	http://www.yuyan-lang.org/
仓库	https://github.com/yuyan-lang/yuyan

2.15.1 简介

本科研性质的项目旨在解决如下问题：

- 一个中文编程语言可以以何种形式存在？
- 编程中概念应当如何用符合中文语境的概念表达？
- 中文语法语义从何种程度上可以帮助程序的撰写？
- 编程语言在全中文环境下会遇到哪些技术问题，它们应该被如何解决？
- 中文的概念和表达又会给编程语言本身带来何种新的发展？

作者相信，编程语言的设计本身并非完全是一个科学问题，更是一个艺术问题。 本项目若可以为朋友们提供一些设计中文编程语言技术、艺术、与文化方面的思路， 我们也就实现了本项目的初衷。

设计理念

豫言中文编程语言以中文编程为核心，以现代化编译器框架 LLVM 为基础，吸取函数式编程领域数十年来的语言设计经验，自主研发，实现了从顶部语法，编译设计，代码生成的全中文编程环境。豫言编译器将全中文的源码，通过一系列编译步骤，生成了完全使用中文标识符 LLVM 后端码，最终由 LLVM 编译器框架生成后端执行程序。豫言编译器本身也

使用了豫言编程语言实现，证明了豫言语言设计可以被用来构建大型程序，是众多编程语言以外企业和个人的又一项选择。

核心特征

与其他语言相比，豫言有着独特的风格。与其余大部分中文编程语言相比，豫言基于函数式编程，开创性地采用了依值类型系统，从根本上增强了语言的安全性与可靠性，也从某种程度上增加了软件开发效率。与同类型的英文编程语言相比，豫言的出现减少了语言学习的门槛，使得广大软件行业从业者乃至青少年不需要借助英文就可以学习和使用先进的编程语言范式，同时这些范式在豫言中拥有更直接的表达，这对于汉语在计算机行业及编程语言相关技术的发展有着直接的促进作用。

历史展望

豫言编程语言不是第一个，也不会是最后一个中文编程语言，我们已经看到有新的中文编程语言（例如入墨答语言）借用了豫言编程语言中的一些设计。我们希望全新设计的豫言编程语言成为集当今优秀的编程语言设计于一体，能够兼顾工业生产、人才教育、科学研究的一门编程语言。未来一定会有新的编程语言出现，我们希望通过豫言成为他们设计时的可靠参考。

为何设计中文编程语言？

中文有源远流长的历史，丰富的文化内涵。这样一门语言，在科技上仍然看不到广泛的使用，定有其特殊的原因。中文编程语言的设计虽然经历了照搬翻译英文关键字，自主设计语法，到使用本土表达的发展阶段，但目前仍有实用方面未能够解决的问题。我希望借本项目，探索中文在编程语言设计领域的各种问题，最终展现中文在编程语言行业的魅力。

附录一 语言类别

a. 一般编程语言（General Programming Languages）

b. 并行语言（Parallel Programming Languages）

c. 并发语言（Concurrent Programming Languages）

d. 分布式语言（Distributed Programming Languages）

e. 命令式语言（Imperative Languages）

f. 面向对象语言（Object Oriented Languages）

g. 函数式语言（Functional Languages）

h. 约束和逻辑语言（Constraint and Logic Languages）

i. 数据流语言（Data Flow Languages）

j. 可扩展语言（Extensible Languages）

k. 汇编语言（Assembly Languages）

l. 多范式语言（Multiparadigm Languages）

m. 高级编程语言（Very High Level Language）

附录二 工具类别

a. 一般编译工具（General Compilers）

b. 解释器（Interpreters）

c. 增量编译器（Incremental Compilers）

d. 可重定向编译器（Retargetable Compilers）

e. 实时编译器（Just-in-time Compilers）

f. 动态编译器（Dynamic Compilers）

g. 生成器（Translator Writing Systems and Compiler Generators）

h. 代码生成（Source Code Generation）

i. 运行时环境（Runtime Environment）

j. 预处理器（Preprocessors）

k. 解析器（Parsers）

附录三 应用领域

a. 通用（General Computation）

b. 计算理论（Theory of Computation）

c. 计算数学（Mathematics of Computing）

d. 网路（Network）

e. 信息系统（Information Systems）

f. 安全（Security）

g. 机器学习（Machine Learning）

h. 人工智能（Artificial Intelligence）

i. 并行计算（Parallel Computing）

j. 并发计算（Concurrent Computing）

k. 分布式计算（Distributed Computing）

l. 建模与模拟（Modeling and Simulation）

m. 计算机图形（Computer Graphics）

n. 行业应用（Applied Computing）