BLUE BOOK ON PROGRAMMING LANGUAGES IN CHINA 2024

Programming Language Open Community（PLOC）
Hubei Software Industry Association

Copyright

- Special Advisers: Yuning Liang, Jianzhong Li
- Curators: Shushan Chai, Ernan Ding, Zhiyong Li, Hailong Yang
- Editors: Chaochen Chen, Dengchun Li, Haokun Li, Sen Wu, Fang Xu, Hailing Yang, Puming Zhao, Zirun Zhu
- Supporting: GitCode
- Sponsor: Wuhan WaYuYan Technology Co., Ltd.

"The Blue Book plays an important part of programming technology evolution in terms of communication among all the enthusiasts, for example between the application developers who using the programming languages and those engineers who making the programming languages. As time goes by, it's a good approach to collect them and debate among them along their strengths and weaknesses which changes against time. For example, in the beginning of computer era, computer scientists are rare who usually know a lot from the fundamentals to application, at that time programming languages were designed for experts. Since Internet era begins, programming languages evolved from a complex version to a naive one like JavaScript which hugely popular for novice programmers who making web applications instead of complex scientific applications. Now AI age starts, we can foresee new programming languages might become for AI models instead of for human! The value of this Blue Book is obvious and becomes more relevant in the future evolution in AI computing. Keep up the valuable work and all the best. "

- Yuning Liang

"Programming languages are the intersection between humans and computers, playing an indispensable role in the entire computing industry. Since the creation of the first advanced programming language, FORTRAN, in 1957, nearly seventy years have passed. During this term, the computer industry has given birth to hundreds of programming languages, with over twenty still in widespread use today. The evolution of programming languages bears a remarkable resemblance to the history of thousands of human languages.

Programming languages are also catalysts for technology revolution. With each new generation of technology revolution, corresponding programming languages emerge or flourish. For instance, C++ for system software, JavaScript for the Web, Java/Go for cloud-native development, and Python for machine learning, among others.

In the era of artificial intelligence driven by large language models, the code auto-generation technology brought about by these models is pioneering a new development paradigm and opening up unprecedented possibilities for the evolution of programming languages.

With the tireless efforts of numerous experts, scholars, and engineers, programming languages from China are emerging and are becoming a vibrant part of the grand vision for the development of domestic software. The compilation of the Blue Book on Programming Languages in China is highly significant for promoting communication and development within this field. Let us join hands to support the development of programming languages in China!"

- Jianzhong Li

Contents

Charter 1 Preface

1.1 Background

Programming language can be considered the mother machine of the software industry, while compiler technology represents the root technology of the IT-industry. A variety of programming languages are employed in the development of operating systems, database management systems, network services, industrial control equipment, applications, and other modern industries and service areas. In particular, the continuous expansion of the innovation space of the information industry, the continuous increase of system complexity, and the continuous reduction of development costs have been directly benefited from the continuous emergence of programming languages and compiler technologies. Currently, there is a notable absence of widely used programming languages in China, which is incongruous with the country's status as a world-class industrial and scientific power.

The Ministry of Industry and Information Technology published the '14th Five-Year Plan' software and information technology services development plan, which states that there should be 'strengthened supply of basic components' and 'accelerated breakthrough in programming language development frameworks'. Chapter 4 of the 'China Software Root Technology Development White Paper (Basic Software Book)', published by the China Software Industry Association (CSIA), is dedicated to summarizing the importance and development trends of programming languages and compilers. The aforementioned documents demonstrate that the advancement of programming language-related industries has been the subject of governmental policy initiatives. Following years of development and accumulation in China, information technology has emerged as a significant economic sector, employing nearly ten million individuals. The necessity for programming languages as a fundamental tool has reached a considerable level of demand. Additionally, the rapid growth observed in emerging domains, such as large language models and domestically manufactured chips, has introduced a multitude of novel requirements for programming languages.

A review of historical data reveals that, in contrast to other industries, the success stories of programming languages, the foundation of the information industry, are characterized by a high degree of serendipity. The current landscape of widely used programming languages and development tools encompasses both commercial projects driven by large enterprises and opensource projects initiated by individuals. These can be classified as either KPI-driven commercial products or interest-driven products. Additionally, the domestic software industry is witnessing a trend of highly fragmented projects, with a considerable number of emerging programming language projects of diverse types and for various domains initiated by enterprises and opensource communities.

1.2 Our Purpose

Based on the background above, PLOC initiated the compilation and publication of the "Blue Book on Programming Languages in China" (hereinafter referred to as the Blue Book), striving to comprehensively collect active programming language projects that have certain usability in China, and provide the industry with an objective panoramic view of domestic languages. We hope that the Blue Book will reflect the overall situation of domestic programming language projects as objectively as possible, provide a global perspective for the industry, assist the industry's demand side in finding suitable languages, and help programming language enthusiasts find opensource projects to contribute to. The Blue Book will be released regularly to track the latest developments in the industry.

"Practitioners supporting each other" is the core concept of PLOC, and the Blue Book inherits this feature. The projects included in this book are all self-reported, and the editor will review the project qualifications; the project content (text, pictures, etc.) is provided by the project author, and the editor only adjusts the page layout. The person who knows the characteristics of the language best is the language author. We hope that through self-declaration, the characteristics of each project can be presented in the form that best suits the author's personality, in order to attract like-minded enthusiasts, contributors, and potential users.

In order to keep the information up to date, the Blue Book will continue to be updated and released. "Chinese Programming Language Blue Book 2024" is the second edition of the Blue Book. Based on the experience of compiling the 2023 edition and feedback from all parties, the 2024 edition has the following updates:

- Added English version. The Chinese and English versions will be published separately with consistent content;
- A list of recommended contents has been added to the project profile section of the application materials to facilitate the standardization of project presentation.

1.3 Inclusion Criteria

Projects that meet the following conditions can be submitted through PR in the Blue Book Workspace Repository:

1. The project is initiated and maintained by an enterprise, community or individual in China;
2. The project meets the project classification standards (see Section 1.5);
3. The project is basically available and can be independently verified by the editorial board;
4. It is open to the public;

5. The project is active.

The editorial board of Blue Book has the final right of interpretation of the inclusion criteria.

1.4 How to submit

All projects included in the Blue Book are self-reported. Projects that meet the inclusion criteria can be submitted at the following address:

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

Add a directory with the same name as the project in the above directory, and fill in the project introduction and other information in markdown format. The information should include the following:

- Project name
- Project icon
- Project homepage
- Project warehouse
- Project classification label
- Project introduction in Chinese and English
- Applicant contact information

The project introduction section is recommended to include the following content, in any format:

- Target application scenario
- Project features and design concept
- Examples (such as "hello world")
- Project goals etc.

For projects that do not provide an English version of the introduction, the editorial board will translate it independently and incorporate it into the English version of the Blue Book. Since it is impossible to guarantee that the translation accurately expresses the content of the project, please provide a bilingual version with consistent content as much as possible. No more than 4 pictures can be inserted in both the Chinese and English versions. The picture size is 1920×1080 pixels (20

×11.25 inches, 96ppi), and the picture format can be JPG or PNG. See the application example:

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

The Blue Book will use font size 10.5 and vertical A4 layout, with the Chinese and English versions in separate volumes. The total length of the project information pages for the Chinese version should not exceed 4 pages, and the English version should not exceed 6 pages. The final sample layout is available at:

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

After you submit your PR, the editorial board will review the project information. During this period, please keep the project address and website accessible. The editors will contact you to confirm that the project information is accurate. If you have any questions about how to fill in certain options, you can also communicate with them at this time. By submitting a project, you are deemed to have received permission from the project owner and authorise the Programming Languages Open Community (PLOC) to display the project name, icon and other information in the Blue Book.

1.5 Project classification

Language project classification tags:

- Commercial / Free
- Opensource / Closed source
- General / DSL
- Accepting Community Contribution or not
- Language tags (see Appendix - Language Category List)
- Tool tags (see Appendix - Tool Category list)
- Application areas (see Appendix - Application Area list)

Tool project classification tags:

- Commercial / Free
- Opensource / Closed source
- Accepting Community Contribution or not
- Tool tags (see Appendix - Tool Category list)
- Application areas (see Appendix - Application Area list)

Charter 2 Summary of 2024

Compared with the 2023 edition, the number of projects included in 2024 edition has increased from 15 to 20, and the changes are as follows:

- 6 new projects, namely: GödelScript, Goldfish Scheme, MoonBit, Nasal-Interpreter, PikaPython, XLang;
- 1 project was removed: Yuyan (withdrew by the project owner);
- 2 renamed projects: Z Lang was renamed Auto Lang, K Lang was renamed Koral

From January 1, 2024 to the time of writing, the update status of each project is shown as follow:

Project Name	Revisions	Project Name	Revisions
Auto Lang	155	Aya	639
Calcit	168	CovScript	14
DeepLang	23	GödelScript	7
HVML	32	Goldfish Scheme	244
KCL	505	Koral	25
Losu	73	MoonBit	1550
Nasal-Interpreter	137	NASL	2609
PikaPython	206	Qing	27
RoarLang	178	TuLang	324
Wa	506	XLang	1256

Since each project follows different rules (such as whether to use Squash, etc.), the above data only indicates whether the project is active and should not be compared with each other.

In 2024, programming language community activities in China have made great progress, many activities organized by opensource communities have opened programming language sub-forums for the first time, including:

- In March 2024, the 11th OS2ATC, programming language sub-forum, with Yongming Wei (member of CPLOC) as the producer:

Fig. 2-1

- In April 2024, PLOC hold its first meetup in Hangzhou:

Fig. 2-2

- In October 2024, PLOC participated in the G-Star Carnival and won the "GitCode Top Ten Opensource Communities of the Year" award:

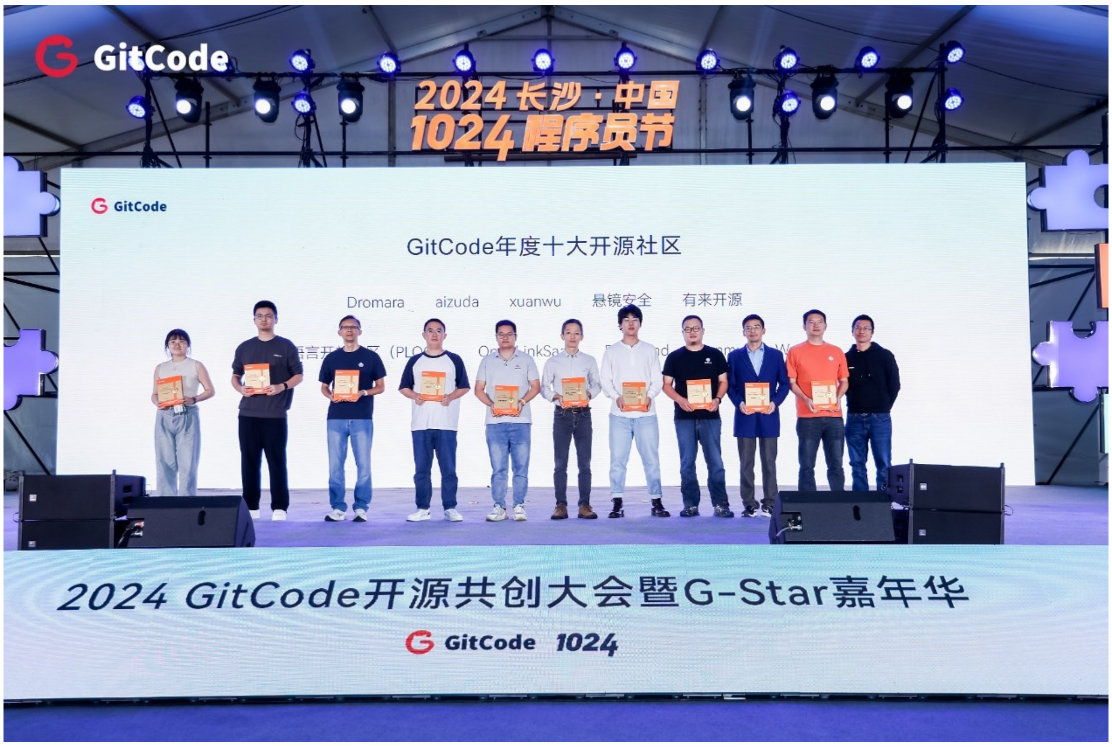


Fig. 2-3

- Several CPLOC member projects were included in the "G-Star Landscape":

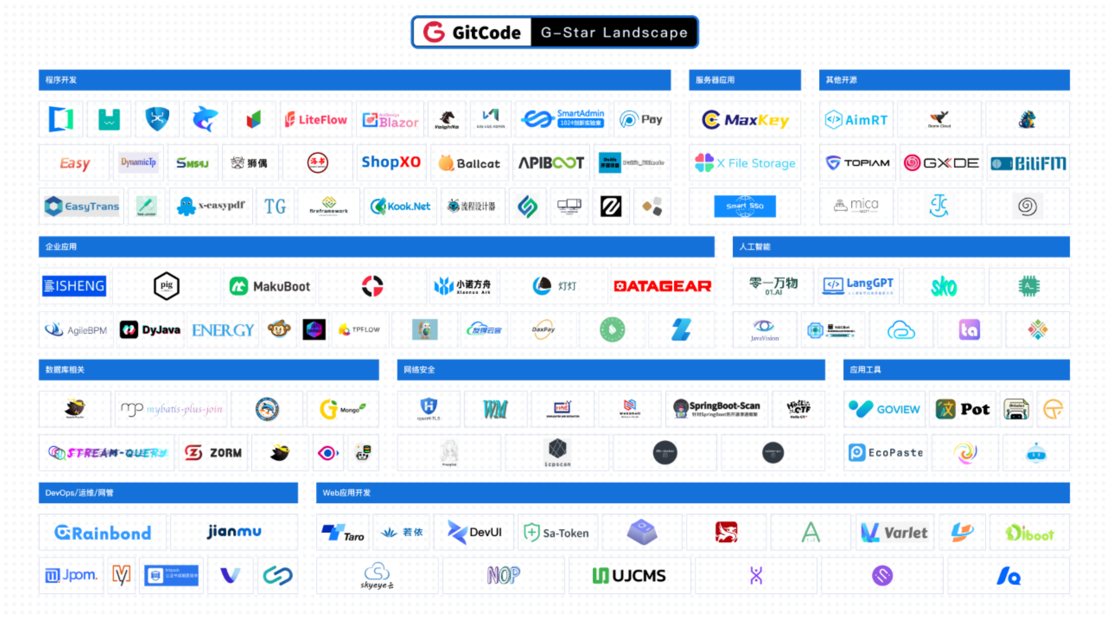


Fig. 2-4

- In November 2024, the COSCon'2024, programming language sub-forum, jointly produced by KAIYUANSHE and PLOC, with Ernan Ding (Secretary of CPLOC) as the producer:

Fig. 2-5

This series of activities shows that the importance of programming language as the mother machine of software industry is increasingly valued by all walks of life. We are sincerely pleased that PLOC can participate in this process and make its own contribution!

Charter 3 Project List

A total of 20 projects are included in the Blue Book, and the names of each project are as follows (in alphabetical order, regardless of Chinese and English):

- Auto Lang
- Aya
- Calcit
- CovScript
- DeepLang
- GödelScript
- HVML
- Goldfish Scheme
- KCL
- Koral
- Losu
- MoonBit
- Nasal-Interpreter
- NASL
- PikaPython
- Qing
- RoarLang
- TuLang
- Wa
- XLang

3.1 Auto Lang

AUTO LANG

Project Tags	Language, Free, Opensource (MIT), General, Accepting Contribution
Language Tags	General
Tool Tags	Interpreter, Source Code Generation(C, Rust, Python)
Application Areas	Industry Application(Automotive, Embedded, Robotics, UI), Computer Graphic
Homepage	https://gitee.com/auto-stack/auto-lang
Repository	https://gitee.com/auto-stack/auto-lang

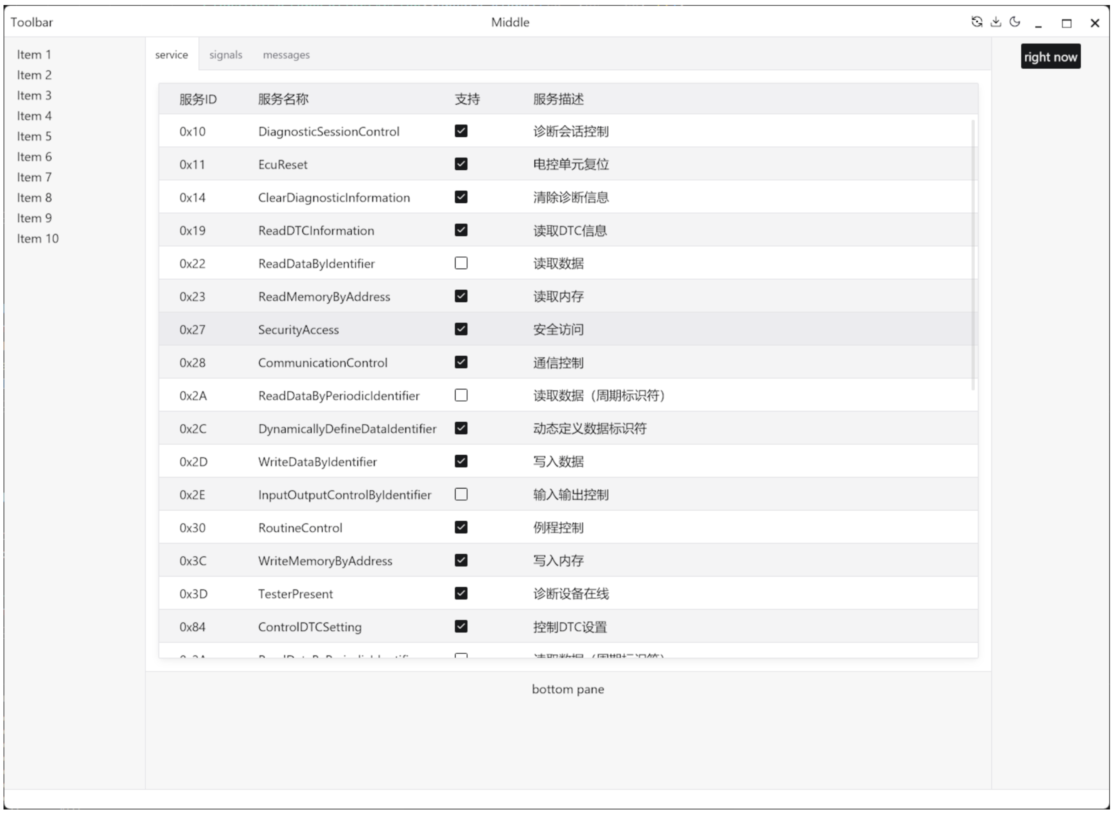
3.1.1 Introduction

The Auto language is a programming language for multiple scenarios. It is implemented based on Rust. Features including:

- Flexibility. Adapt to multiple ecosystems (C, Rust, Python, Shell, etc.). Auto provides multiple syntax, language features and standard libraries for different scenarios;
- Static and dynamic typing. Interpretation and static compilation;
- Full Stack. Auto has its own standard library, REPL, builder and UI framework.

The Auto language can be used in the following scenarios:

- AutoUI: As the description language for UI.



服务ID	服务名称	支持	服务描述
0x10	DiagnosticSessionControl	☑	诊断会话控制
0x11	EcuReset	☑	电控单元复位
0x14	ClearDiagnosticInformation	☑	清除诊断信息
0x19	ReadDTCInformation	☑	读取DTC信息
0x22	ReadDataByIdentifier	☐	读取数据
0x23	ReadMemoryByAddress	☑	读取内存
0x27	SecurityAccess	☑	安全访问
0x28	CommunicationControl	☑	通信控制
0x2A	ReadDataByPeriodicIdentifier	☐	读取数据（周期标识符）
0x2C	DynamicallyDefineDataIdentifier	☑	动态定义数据标识符
0x2D	WriteDataByIdentifier	☑	写入数据
0x2E	InputOutputControlByIdentifier	☐	输入输出控制
0x30	RoutineControl	☑	例程控制
0x3C	WriteMemoryByAddress	☑	写入内存
0x3D	TesterPresent	☑	诊断设备在线
0x84	ControlDTCSetting	☑	控制DTC设置

Fig. 3.1-1

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

- AutoMan: Manage mixed projects in Auto/C as a configuration language.

```
project: "hello"
version: "0.1.0"
// Dependencies
dep(log, "0.1.0")
// Libs
lib("mymath") { link: log }
// Executables
exe("hello") { link: mymath }
```

- Translation to C and management of hybrid Auto/C projects using AutoMan tools.

```
// 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;
}
```

- as Shell script for cross-platform scripting functionality.

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

- as template for code generation in C/Rust/HTML, as follows:

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

This loop is expanded with data in students filled into each row, and generated into full HTML code.

- as embedded scripts to assist Rust development. For example, in the upcoming project AutoEngine, we use Bevy as the underlying 3D graphics engine, and Auto as the scripting language to manage game logics.

3.2 Aya

Project Tags	Language, Free, Opensource (MIT), General, Accepting Contribution
Language Tags	Functional Language
Tool Tags	Interpreter, Just-in-time Compiler
Application Areas	General, Mathematics of Computing, Applied Computing (Compiler Development)
Homepage	https://www.aya-prover.org
Repository	https://github.com/aya-prover/aya-dev

3.2.1 Introduction

The Aya language is a functional programming language similar to Haskell and Lean4 with language features such as inductive types, pattern matching, and first-class functions as the main code organization tools.

Aya has a more powerful type system than Haskell, with support for dependent types and equality types, and the equality has better properties compared to Lean4. In Aya, two values are equal is a type, and its instance is the proof witness that these two values are equal. For example, insertionSort = mergeSort is a valid type, and in Aya it is directly equivalent to the function (x : list) -> insertionSort(x) = mergeSort(x), and an instance of which needs to take a list, and return a proof that it is the same after being sorted by either sorting algorithms. Such a proof can be used to prove some properties about the program while programming.

For a more detailed description of the project's motivation in the Aya team's recruitment post:

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

For academic papers on language features in Aya:

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

3.3 Calcit

Project Tags	Language, Free, Opensource (MIT), General, Accepting Contribution
Language Tags	General (Scripting Language)
Tool Tags	Source Code Generation (JavaScript), Interpreter
Application Areas	Applied Computing (Web Development)
Homepage	https://calcit-lang.org/
Repository	https://github.com/calcit-lang/calcit

3.3.1 Introduction

Calcit is a dialect of Clojure, following the core design of immutable data structures, prefix expressions, and Macros. Implemented in Rust for fast startup and runtime, Calcit can be interpreted and executed directly, or compiled to JavaScript code for execution.The generated code is simplified with ES Modules and other modern front-end development habits, which makes it lighter than the ClojureScript solution, easier to mix and match with JavaScript code, and reduces debugging costs to a certain extent.Calcit's text forms use the indentation syntax.

Code example one, a simple data transformation, similar to Clojure syntax in the threading macros:

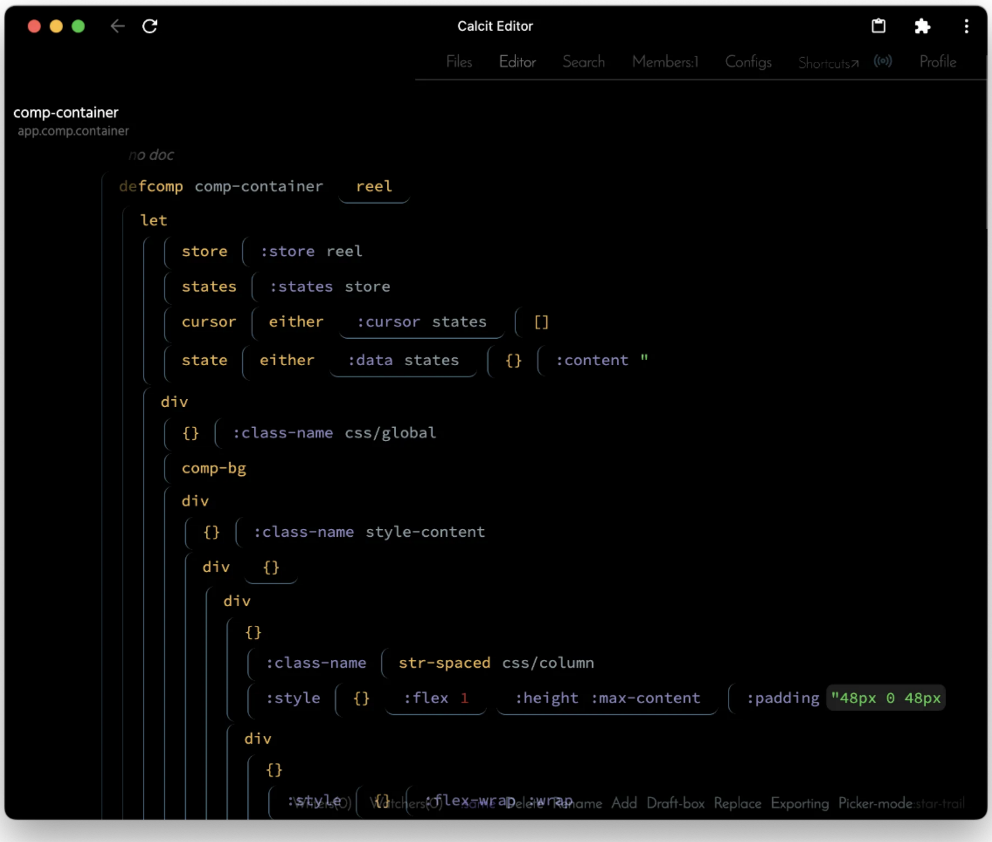
```
->
  range 100
  map $ fn (x)
    * x x
  foldl 0 &+
  println
```

Code example two, using Macro-wrapped front-end Virtual DOM component writing based on the Calcit eco-definition:

```
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
```

In practice, Calcit uses data files to store source code. Support editing expressions directly in a structured way, with the editor unfolding in the form of data, thus also allowing for rapid partial definition adjustments and code restructuring, thus increasing the speed of writing and modifying dynamically typed languages:



```
defcomp comp-container reel
  let
      store (:store reel)
      states (:states store)
      cursor (either (:cursor states) [])
      state (either (:data states) ({} (:content "))
    div
      {} (:class-name css/global)
      comp-bg
      div
        {} (:class-name style-content)
        div {}
          div
            {}
              :class-name (str-spaced css/column)
              :style ({} (:flex 1) (:height :max-content) (:padding "48px 0 48px
            div
              {}
                :style ({} (:flex-wrap :wrap
```

Fig. 3.3-1

Calcit is mainly used in web page development scenarios. Implemented some of the features of the Virtual DOM ecosystem.

3.4 CovScript

Covariant Script

Never cease the pace of seek

Project Tags	Language, Free, Opensource(Apache2.0),General, Accepting Contribution
Language Tags	Imperative Language
Tool Tags	Interpreter, Just-in-time Compiler, Runtime Environment
Application Areas	General
Homepage	https://covscript.org.cn
Repository	https://github.com/covscript

3.4.1 Introduction

The Covariant Script programming language, CovScript for short, or 智锐编程语言 for short in Chinese, originally released in 2017, is a cross-platform, open-source, dynamically-typed, application-layer general-purpose programming language that is efficient, easy to learn, easy to use, and reliable, combining the advantages of modern programming languages, and can efficiently and directly interact with C++ via CNI It is a cross-platform, open source, dynamically typed application layer general purpose programming language.

CovScript programming language is one of the first batch of independent intellectual property programming languages put on the market in China, with an independent and perfect tool chain, including basic Interpreter, debugger, Just-in-time Compiler (JIT Compiler), standard libraries, extension libraries, documentation and IDE plug-ins, etc., which is not dependent on the existing programming language Runtime. CovScript's autonomous, independent, complete and reliable language and attached ecosystem have made CovScript widely praised by customers, and it has already been deployed in Sichuan University's Information Technology Construction and Management Office, Sichuan University's Huaxi Big Data Centre, etc., providing 7x24-hour services in critical systems.

```
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
```

The CovScript programming language is a multi-paradigm programming language that is primarily imperative, supplemented by object-oriented and functional programming, and is easy to understand and intuitive for beginners, as well as for solving the needs of large-scale projects. Currently there are several mature development frameworks for CovScript:

- CovAnalysis: a data analysis and processing framework that outperforms Pandas.
- CSDBC: ODBC-based database connectivity, compatible with most major RDBMSs.
- ParserGen: a real-time grammar parser generator based on EBNF-like rules, based on which CovScript implements a fully bootstrapped.

In addition, CovScript has a comprehensive package manager and a myriad of tools to assist in the development of most cloud-native applications.

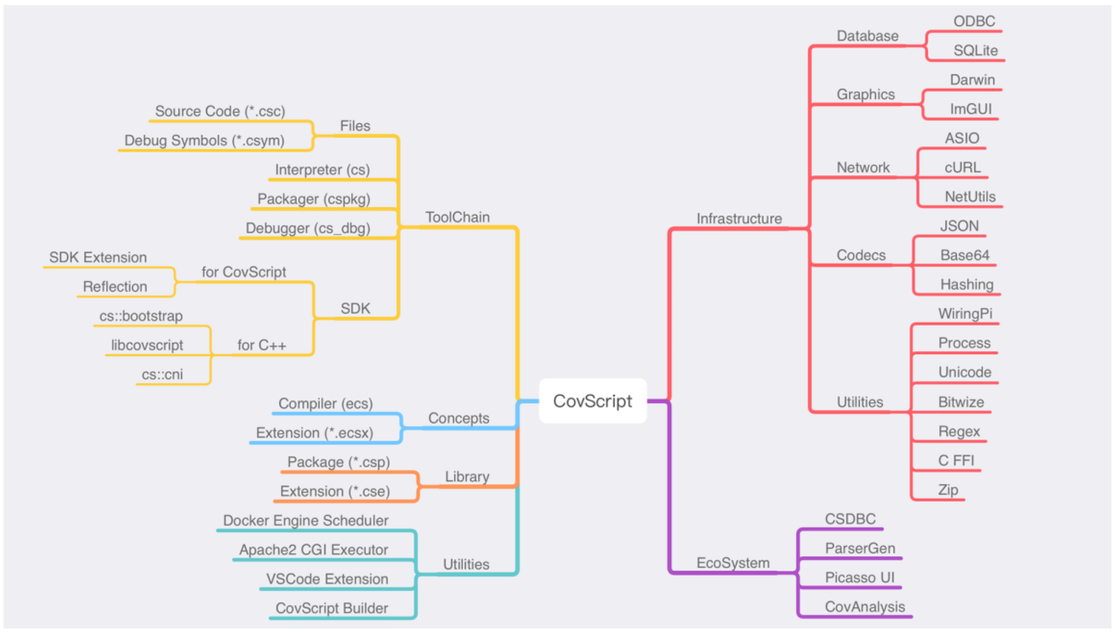


Fig. 3.4-1

Although CovScript is a dynamic programming language, its core runtime is written in highly optimized C++ code, with an execution speed of up to 9 million lines of code per second, and a context switching speed of 80 GOPS (billions of operations per second) for concurrent processes, which enables it to efficiently support a wide range of application requirements.

As a programming language designed and developed by Chinese people, CovScript is practicing Made in China with its own actions. The core ecology of the language has 100% independent intellectual property rights (registered in the National Copyright Administration of the People's Republic of China, Registration No. 2020SR0408026; retrieved by Zenodo, DOI No. 10.5281/zenodo.10471188), and the surrounding ecology is 100% Opensource and trustworthy. Zenodo search, DOI number: 10.5281/zenodo.10471188), the surrounding ecology is 100% Opensource, credible. Not only that, CovScript also fully supports domestic ecology:

- specially optimized and tested for the Loongson architecture and Chinese operating systems.
- Each CovScript distribution will have a corresponding Dragoncore version (UOS@3A4000).
- Compatible with Huawei Kunpeng processor and OpenEuler OS.
- CSDBC Compatible with Huawei OpenGauss Database.

In order to serve as many customers as possible, CovScript is also compatible with stock systems. In addition to the mainstream version being compatible with Windows 7 64bit, it can also be customized to be compatible with Windows XP SP2.

The years 2017 to 2022 were five years of rapid development that gave CovScript a very mature ecology comparable to that of mature programming languages. Since 2023, CovScript began planning its move into modern programming languages and cutting-edge Application Fields, first by refining the fourth generation of the language standard (CovScript 4, or ECS), and then verifying the viability of the CovTorchMachine Learning framework. In the future, CovScript will focus on building the critical infrastructure in the LLMOps process to empower cutting-edge applications.

3.5 DeepLang

Project Tags	Language, Free, Opensource (MIT), General, Accepting Contribution
Language Tags	General
Tool Tags	General
Application Areas	General
Homepage	https://deeplang.org/
Repository	https://github.com/deeplang-org/deeplang

3.5.1 Introduction

DeepLang is a programming language designed for resource-constrained scenarios, with static typed and strongly typed features, a syntax style referring to C-style design. And DeepLang supports a hybrid programming paradigm of procedural, logical, and functional programming. For Internet of Things (IoT) applications, DeepLang has various memory safety features, and its design draws on the safety mechanisms of Rust and selects more appropriate methods of compilation and execution for resource-constrained scenarios.

DeepLang's tool set consists of a compiler, Deepc, and a virtual machine, DeepVM. Deepc is implemented by OCaml and has a multi-stage code processing flow: firstly, a grammar tree is generated by a parser, then a walker is used to traverse the tree to construct a symbol table. The conversion module translates the tree into ANF IR, and finally converts ANF IR into WASM bytecode through CodeGen. Currently, Deepc only supports type checker and has no type infer, so all DeepLang source code must be explicitly annotated with types, otherwise it will be regarded as a syntax error. DeepVM is implemented by C and supports WASM 1.0, with features such as bytecode loading, memory management, interpreted execution, and FFI.

Currently, the DeepLang team consists of Masters and PhDs from Jiangnan University, Imperial College London, Zhejiang University, and University of Science and Technology of China. And our team focuses on the design of language features in resource-constrained scenarios. Due to the limited energy and resources of our team, DeepLang does not have any commercialization capability at this stage.

Example of DeepLang's ADT feature:

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

Examples of DeepLang's interface feature:

```
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
}
```

Examples of DeepLang's Pattern Matching Feature:

```
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

Project Tags	Language, Free, Opensource (Apache2.0), DSL, Accepting Contribution
Language Tags	Domain Specific Language, Declarative Language
Tool Tags	General, Source Code Generation, Interpreter
Application Areas	Industry Applications(Code Static Analysis, Database)
Homepage	https://github.com/codefuse-ai/CodeFuse-Query/blob/main/godel-script/README.md
Repository	https://github.com/codefuse-ai/CodeFuse-Query/tree/main/godel-script

3.6.1 Introduction

GödelScript was designed by the original Ant Group CodeInsight team: Chen Xinyu, Fan Gang, Fu Jinjian, Liang Yinan, Li Haokun, Li Shijie, Shi Qingkai, Wang Wenyang, Xiao Ling, Zhou Jinguo, and Zhen Yi (in alphabetical order).

The "hello world" code in GödelScript is as follows:

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

GödelScript is CodeQuery's domain-specific language (DSL) for querying and data processing. The underlying engine is Soufflé Datalog. GödelScript uses a Rust-like syntax that provides strict type-checking, fast and easy type derivation, and smart and friendly error messages to get you started.

The main application scenarios for the GödelScript compiler as follow:

- User-oriented writing of simple or complex queries, providing a more convenient way to write queries and improving the efficiency of query writing.
- provides strict type checking and type derivation, giving smarter code modification hints.

- provides strict ungrounded detection to avoid triggering the Soufflé Ungrounded Error;
- Language Server and IDE Extension.

Since it is a DSL (Domain Specific Language) based on Datalog, the language offers a number of Datalog-specific features such as:

```
// 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();
    }
}
```

And the function to get the query result can be written as:

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

Or use SQL-like syntax:

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

3.7 HVML

Project Tags	Language, Free, Opensource(Multiple License), General, Accepting Contribution
Language Tags	General
Tool Tags	General, Runtime Environment
Application Areas	General
Homepage	https://hvml.fmsoft.cn/
Repository	https://github.com/HVML

3.7.1 Introduction

HVML stands for Hybrid Virtual Markup Language. It organizes the presentation of code by means of a markup language; Virtual means that the markup language becomes an abstracted virtual markup language by giving it programming power; Hybrid means hybrid, which is able to organize different languages or programs by means of a glue.

The HVML programming language made the first draft specification publicly available in July 2020; development of the HVML Interpreter began in July 2021; on 30 May 2022, a preliminary version of the HVML graphical renderer, xGUI Pro, was completed; on 3 July 2022, the HVML 1.0 Interpreter PurC, the renderer xGUI Pro stabilized, and we made public all HVML-related source code repositories (or packages); in December 2023, the development of the HVML renderer xGUI was initiated. xGUI will use a self-developed rendering engine, and will be made public in December 2024.

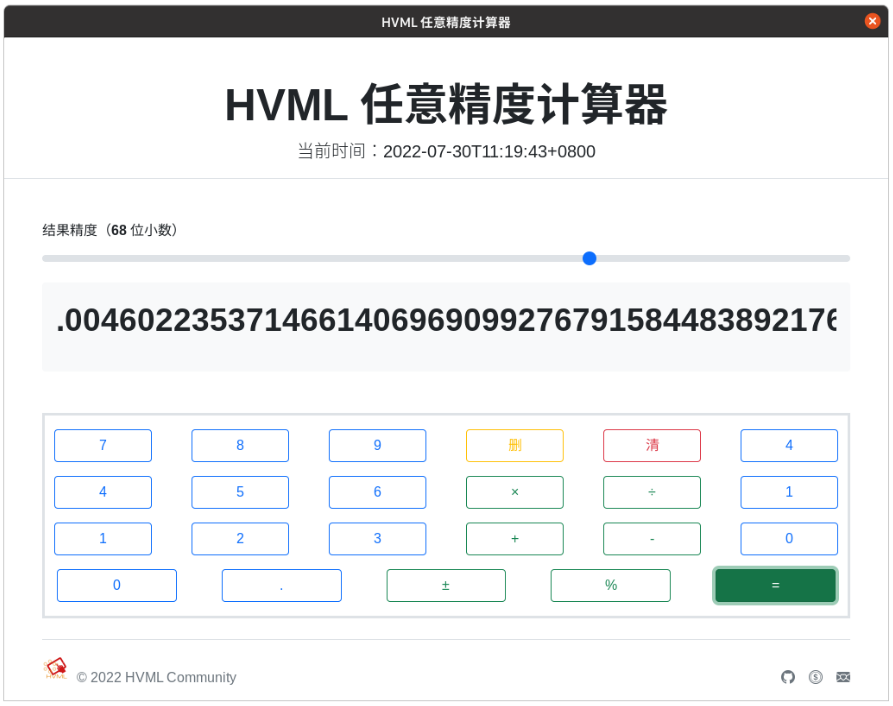
The basic design goal of HVML is to rapidly develop GUI applications using modern web front-end technologies (HTML/SVG, DOM, CSS, etc.) in an existing native runtime constructed in programming languages such as C/C++, Python, etc. without the need for additional browsers or JavaScript engines.

Descriptiveness is a feature of HVML. Descriptive language not only makes it easier for developers to understand and write code, it is also suitable for AI programs for learning and Code Generation. The "hello world" code in HVML is as follows:

```
<!--
   $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 makes it easy to interact with other programs, for example, with the high-precision calculator bc for a graphical interface version of the high-precision calculator.


HVML 任意精度计算器

当前时间：2022-07-30T11:19:43+0800

结果精度（68 位小数）

.004602235371466140696909927679158448389217€

© 2022 HVML Community

Fig. 3.7-1

As well as the ability to embed python code, data interaction with python programs, processing and displaying:

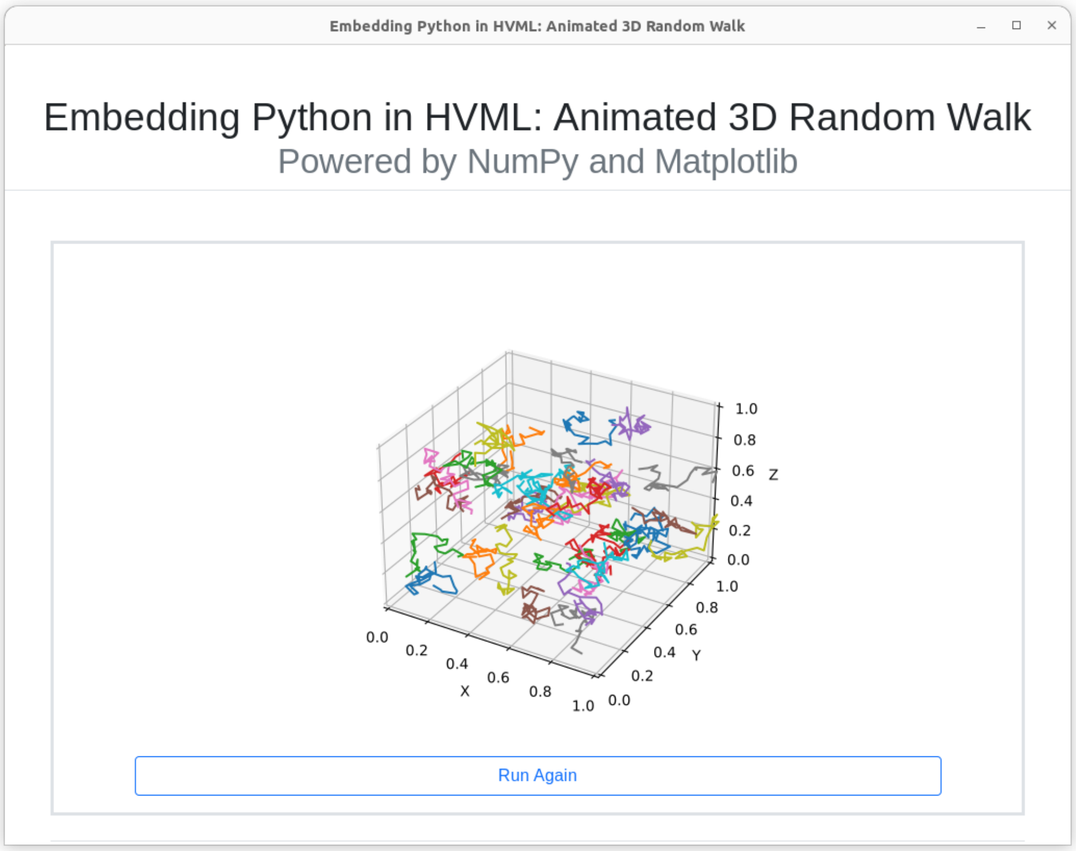

Embedding Python in HVML: Animated 3D Random Walk
Embedding Python in HVML: Animated 3D Random Walk
Powered by NumPy and Matplotlib
X
Y
Z
0.0
0.2
0.4
0.6
0.8
1.0
Run Again

Fig. 3.7-2

As a markup language, the introduction of markup symbols results in a higher number of code characters than in other languages, but the benefit is that the organization of the code appears clearer. This is because another of its design goals is to develop programs with the help of automated graphical low-code development tools, and the clear organization also facilitates access to AI applications.

3.8 Goldfish Scheme

Make Scheme as easy to use and practical as Python
金鱼 Scheme

Project Tags	Language, Free, Opensource(Apache2.0),General, Accepting Contribution
Language Tags	General, Functional Language
Tool Tags	Interpreter
Application Areas	Applied Computing(Education, scientific research)
Homepage	https://gitee.com/LiiiLabs/goldfish
Repository	https://gitee.com/LiiiLabs/goldfish

3.8.1 Introduction

Goldfish Scheme is a Scheme Interpreter with the following features:

- Compatible with R7RS-small Standard
- provides a Python-like standard library.
- Small and fast

Goldfish Scheme was designed with the goal of making Scheme as easy to use and practical as Python.

Goldfish Scheme is an application scenario driven programming language project. As the S7 Scheme used in Mogan STEM Suite does not meet the long-term development of the Mogan STEM Suite, we initiated the Goldfish Scheme project to meet the long-term maintenance of the 76,000 lines of historical Scheme code built into the Mogan, and in addition, plug-ins in the Mogan that are implemented in the Python language, such as the Gnuplot plug-in, we have switched from the Python implementation to the Goldfish Scheme implementation.

Goldfish Scheme is implemented using a Literate Programming approach. We believe that: in the era of LLM, the traditional code-centric programming paradigm will switch to the document-centric Literate Programming paradigm, and the threshold of programming will continue to decrease. Literate Programming as a time-honored programming paradigm will experience a renaissance in the era of LLM.

The documentation for Goldfish Scheme is Chinese-language first, as the originator is a native Chinese speaker. Community communication other than documentation, such as code submission information, code merge request titles and content, and community developers' Lark Suite groups are encouraged to be in English.

3.9 KCL

Project Tags	Language, Free, Opensource (Apache2.0), DSL, Accepting Contribution
Language Tags	Domain Specific Language, Declarative Language
Tool Tags	General
Application Areas	Applied Computing(Cloud Native, Data Engineering, Platform Engineering, etc.)
Homepage	https://kcl-lang.io/
Repository	https://github.com/kcl-lang/kcl

3.9.1 Introduction

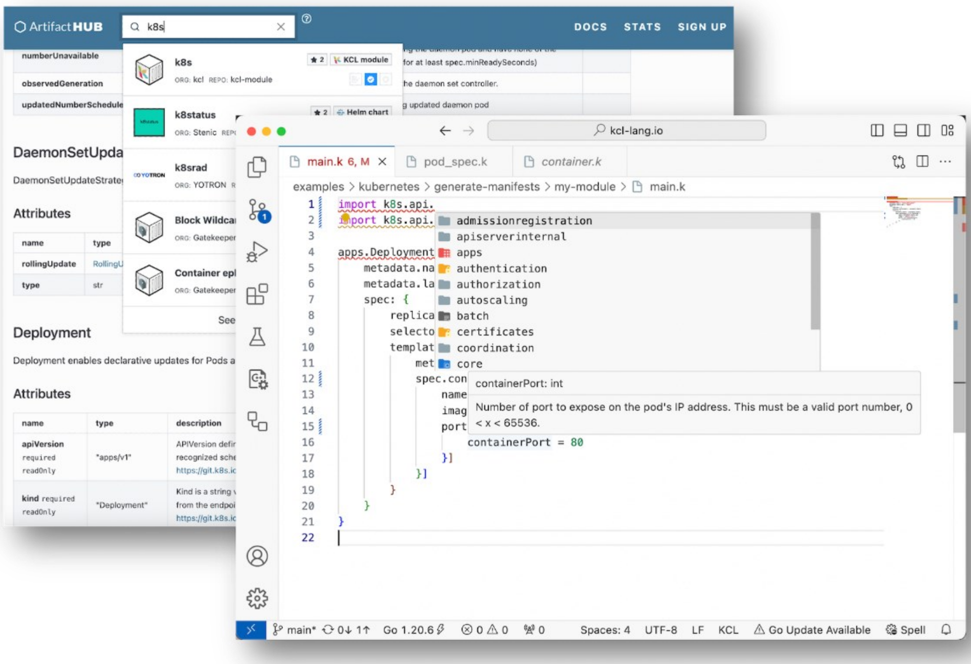
KCL is an opensource constraint-based recording and function language. KCL improves the writing of a large number of complex configurations such as cloud-native scenarios through proven programming language techniques and practices, and is committed to building better modularity, extensibility, and stability around configurations, simpler logic writing, as well as faster automated integrations and good ecological extensibility.

Since it was made available in 2022, it has been adopted and put into products used by many corporate entities and individuals around the world, and in September 2023 it was officially donated to the CNCF Foundation.

KCL can be applied in the fields as follow:

- Generate static configuration data such as JSON, YAML, etc., or integrate with existing data;
- Modelling of configuration data using schema and reduction of sample files in configuration data;
- Define schema with rule constraints for configuration data and automate data validation;
- Organize, simplify, unify and manage large configurations without side effects through a gradient automation program;

- Scalable management of large configurations by writing configuration data in chunks.



name	type
numberUnavailable	
observedGeneration	
updatedNumberSchedule	

DaemonSetUpda

DaemonSetUpdateStrate

Attributes

name	type
rollingUpdate	RollingU
type	str

Deployment

Deployment enables declarative updates for Pods a

Attributes

name	type	description
apiVersion required readOnly	"apps/v1"	APIVersion defir recognized sche https://git.k8s.ic
kind required readOnly	"Deployment"	Kind is a string v from the endpoi https://git.k8s.ic

for at least spec.minReadySeconds)

he daemon set controller.

g updated daemon pod

```
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.

Fig. 3.9-1

3.10 Koral

Project Tags	Language, Free, Opensource (MIT), General, Accepting Contribution
Language Tags	General
Tool Tags	General
Application Areas	General
Repository	https://github.com/kulics/koral

3.10.1 Introduction

The Koral language is an opensource programming language for application domains. It is statically typed, memory-hosted, and multi-paradigm. At this stage, the main goal of the Koral language is to explore type systems and syntax design, and it does not yet have any commercial capabilities or promise any stability.

The Koral language is currently experimenting with a few more interesting designs:

- Generic syntax by case-sensitivity;
- Expression structure syntax based on semicolon and block distinctions;
- Parameterizable variable type qualifiers.

Example1:

```
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
}
```

Example 2:

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

Example 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 Losu

洛书

打造国产编程语言

Project Tags	Language, Free, Opensource (MIT), General, Accepting Contribution
Language Tags	General
Tool Tags	Interpreter, Runtime Environment
Application Areas	General, Industry Application
Homepage	https://losu.tech
Repository	https://gitee.com/chen-chaochen/lpk

3.11.1 Introduction

Losu (Losu: Language of System Units, also known as Easylosu, Losuscript) is an ultra-lightweight cross-platform scripting language designed for low-resource device control and light business development in IoT scenarios, with a minimum resource requirement of only RAM ⩾ 4KB and code space ⩾ 32 KB, aiming to provide It is designed to provide lightweight and dynamic scripting capabilities to improve project flexibility, scalability and customization.

Losu is an innovative programming language with a clean, efficient and reliable design and implementation:

- The main syntax of Losu is in a Python-like style, with simple rules that make it quick to get started:

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

- Losu supports native support for JSON format, efficient data exchange and configuration experience:

```
# json support
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()"}
        ]
    }
}
```

- Losu supports multi-paradigm programming, advanced features such as closures, higher-order functions, duck types, operator overloading:

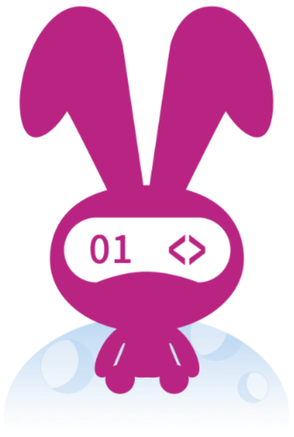
```
# closure
def outer_func(val):
    return def():
        print(val)
var f = outer_func(0)
f()
```

- Losu has good concurrent ability: Losu abandoned the traditional GIL program, their own design and implementation of the C (concatenation) / T (thread) / P (process) three-level scheduling model. Losu concatenation is a native data structure, can be used in a bare-metal environment:

```
# coroutine
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
```

Losu has already released the Minimum Viable Product (MVP) in June 2023, which implements a complete syntax design, compiler, virtual machine, core library, extension library, and implements tools such as package manager and Playground based on itself, initially becoming an emerging programming language with practicality. At the same time, Losu is expected to release the first Minimum Marketable Product (MMP) with corresponding SDK components in early 2025.

3.12 MoonBit



Project Tags	Language, Free, Opensource, General, Accepting Contribution
Language Tags	General
Tool Tags	General
Application Areas	General
Homepage	https://www.moonbitlang.cn
Repository	https://github.com/moonbitlang/core

3.12.1 Introduction

MoonBit is a programming language with a syntax similar to Rust, having GC and coming with modern toolchains and multi-backends:

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

Main advantages:

- Excellent compilation and build speed.
- Simple and practical data-oriented language design.
- Multi-backends: including WebAssembly, WebAssembly with GC, JavaScript and C.
- Generated WebAssembly is small in size and fast in execution; generated C code is efficient.

Features:

- Cloud-native development: MoonBit comes with a cloud-native IDE, which has features for modern IDEs and enables developing, executing, testing, and debugging in browsers without any backends.
- Overall design: MoonBit is designed with the whole toolchain in mind, including

components such as IDE, compiler, and language server, as well as processes such as developing, testing, debugging, and publishing.

- AI-oriented design: MoonBit is designed to cooperate better with AI, having a language design that allows better code generation. All the functions and methods are defined at top-level with complete type declaration and structural traits are used, such as:

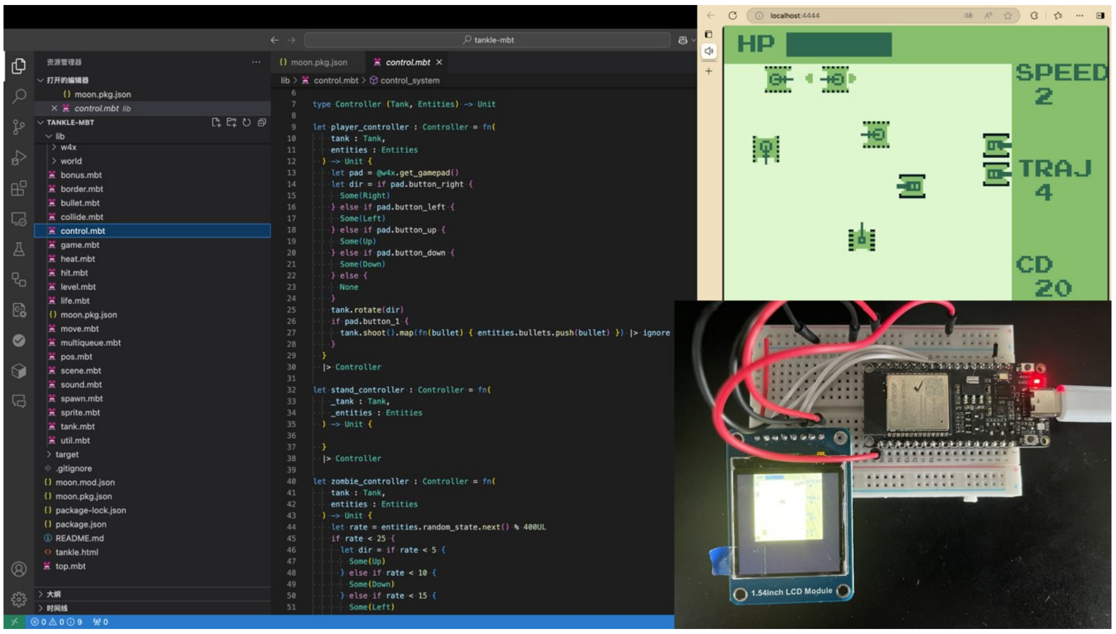
```
pub(open) trait Animal {
  speak(Self) -> Unit
}

struct Dog { }

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

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

- Developer-oriented design: MoonBit support JSON grammar, like this example:

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

Project Showcase:

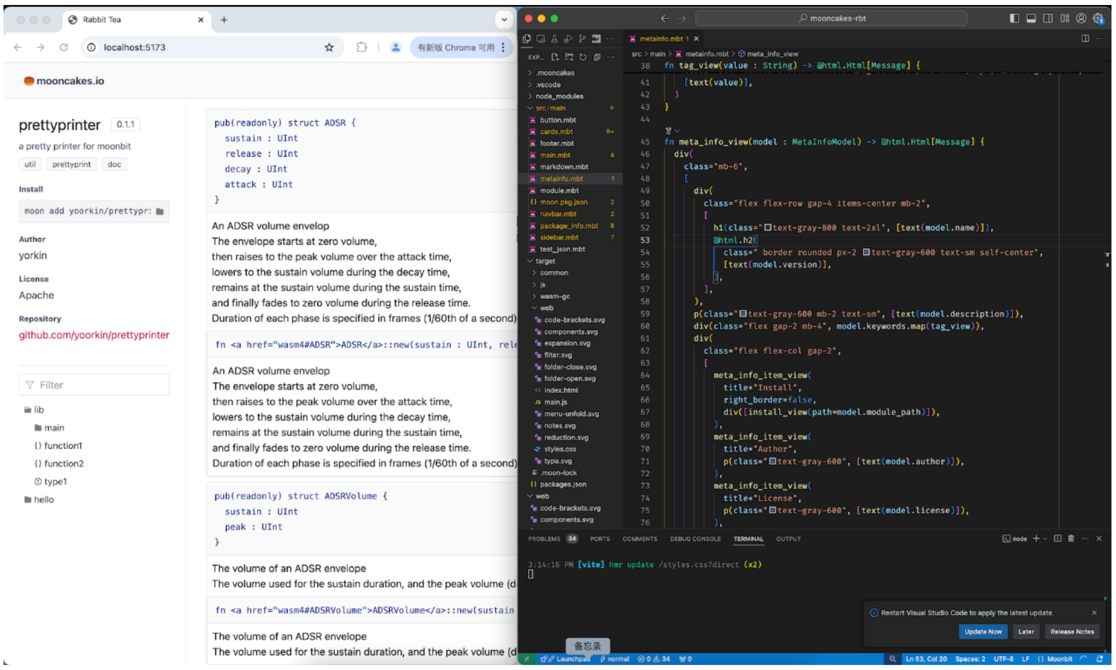
- Developing a game using MoonBit and Wasm4, running in a browser and ESP-C6 microcontroller:



```
type Controller (Tank, Entities) -> Unit

let player_controller : Controller = fn(
    tank : Tank,
    entities : Entities
  ) -> Unit {
    let pad = @w4x.get_gamepad()
    let dir = if pad.button_right {
      Some(Right)
    } else if pad.button_left {
      Some(Left)
    } else if pad.button_up {
      Some(Up)
    } else if pad.button_down {
      Some(Down)
    } else {
      None
    }
    tank.rotate(dir)
    if pad.button_1 {
      tank.shoot().map(fn(bullet) { entities.bullets.push(bullet) }) |> ignore
    }
  }
  |> Controller

let stand_controller : Controller = fn(
    _tank : Tank,
    _entities : Entities
  ) -> Unit {

  }
  |> Controller

let zombie_controller : Controller = fn(
    tank : Tank,
    entities : Entities
  ) -> Unit {
    let rate = entities.random_state.next() % 400UL
    if rate < 25 {
      let dir = if rate < 5 {
        Some(Up)
      } else if rate < 10 {
        Some(Down)
      } else if rate < 15 {
        Some(Left)
```

Fig. 3.12-1

- Web applications developed with MoonBit:



prettyprinter 0.1.1

a pretty printer for moonbit

util prettyprint doc

Install

moon add yoorkin/prettypr

Author

yorkin

License

Apache

Repository

github.com/yoorkin/prettyprinter

```
pub(readonly) struct ADSR {
  sustain : UInt
  release : UInt
  decay : UInt
  attack : UInt
}
```

An ADSR volume envelop
The envelope starts at zero volume,
then raises to the peak volume over the attack time,
lowers to the sustain volume during the decay time,
remains at the sustain volume during the sustain time,
and finally fades to zero volume during the release time.
Duration of each phase is specified in frames (1/60th of a second)

```
fn <a href="wasm4#ADSR">ADSR</a>::new(sustain : UInt, rele
```

An ADSR volume envelop
The envelope starts at zero volume,
then raises to the peak volume over the attack time,
lowers to the sustain volume during the decay time,
remains at the sustain volume during the sustain time,
and finally fades to zero volume during the release time.
Duration of each phase is specified in frames (1/60th of a second)

```
pub(readonly) struct ADSRVolume {
  sustain : UInt
  peak : UInt
}
```

The volume of an ADSR envelope
The volume used for the sustain duration, and the peak volume (d

```
fn <a href="wasm4#ADSRVolume">ADSRVolume</a>::new(sustain
```

The volume of an ADSR envelope
The volume used for the sustain duration, and the peak volume (d

```
fn tag_view(value : String) -> @html.Html[Message] {
    [text(value)],
  )
}

fn meta_info_view(model : MetaInfoModel) -> @html.Html[Message] {
  div(
    class="mb-6",
    [
      div(
        class="flex flex-row gap-4 items-center mb-2",
        [
          h1(class="text-gray-800 text-2xl", [text(model.name)]),
          @html.h2(
            class=" border rounded px-2 text-gray-600 text-sm self-center",
            [text(model.version)],
          ),
        ],
      ),
      p(class="text-gray-600 mb-2 text-sm", [text(model.description)]),
      div(class="flex gap-2 mb-4", model.keywords.map(tag_view)),
      div(
        class="flex flex-col gap-2",
        [
          meta_info_item_view(
            title="Install",
            right_border=false,
            div([install_view(path=model.module_path)]),
          ),
          meta_info_item_view(
            title="Author",
            p(class="text-gray-600", [text(model.author)]),
          ),
          meta_info_item_view(
            title="License",
            p(class="text-gray-600", [text(model.license)]),
          ),
```

```
3:14:16 PM [vite] hmr update /styles.css?direct (x2)
```

Restart Visual Studio Code to apply the latest update.

Fig. 3.12-2

3.13 Nasal-Interpreter

Project Tags	Tool, Free, Opensource (GPLv2), General, Accepting Contribution
Tool Tags	Interpreter
Application Areas	General, Industry Application
Homepage	https://www.fgprc.org.cn/nasal_interpreter.html
Repository	https://github.com/ValKmjolnir/Nasal-Interpreter

3.13.1 Introduction

Nasal is a scripting language with a syntax similar to ECMAscript, designed by Andy Ross, and later introduced into the famous opensource flight simulator FlightGear as a scripting language for the development of the FlightGear model. The "hello world" code in Nasal is as follows:

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

Since debugging in FlightGear is inconvenient using the embedded Nasal console window, just to check for syntax errors you have to spend a lot of time opening the software and waiting for it to load before debugging.

Thus, a brand new Nasal Interpreter was born. The original purpose of the project was to help developers check for syntax errors and even runtime errors. In recent iterations, Nasal-Interpreter also supports REPL Interpreter, cross-platform subprocesses, and more detailed error traceback information.

Nasal's base datatypes are also very simple, and complex datatypes can be realized by combinations of the base types:

```
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's functions are actually Lambda's that can be passed as data:

```
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 uses a rather unusual mechanism to simulate inheritance:

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

Nasal also uses two special loop syntaxes to facilitate scripts:

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

3.14 NASL

NASL

Project Tags	Language, Commercial (No time limit for free trail), Closed Source, DSL, Not Accepting Contribution
Language Tags	Domain Specific Language, Extensible Languages
Tool Tags	Source Code Generation, Incremental Compiler
Application Areas	Industry Application (Low-Code Development Platform)
Homepage	https://nasl.codewave.163.com/

3.14.1 Introduction

NASL, full name Next Application Specific Language, is a domain-specific language used to describe Web applications in NetEase CodeWave intelligent development platform (https://sf.163.com/product/lcap?productId=neteasecloud). It contains two main parts: the basic language and a collection of sub-languages for specific areas of Web applications (e.g., data sources, data queries, pages, processes, permissions, etc.) The most important feature of NASL is the use of the visual editor of the CodeWave Intelligent Development Platform to unify the design of all aspects of the Web application's pages, business logic, data, processes, and so on, supplemented by static checking, full-stack debugging, AIGC Code Generation, multi-person collaboration and other functions:

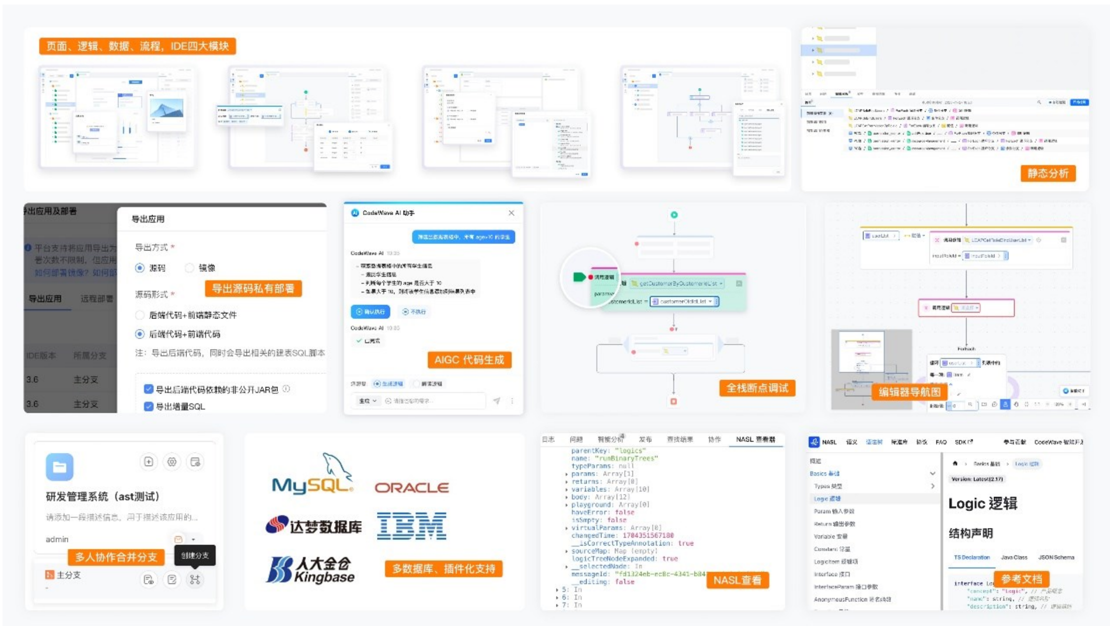


Fig. 3.14-1

For building web applications, NASL and its supporting facilities come out of the box with a low learning threshold and low development costs: developers do not need to learn multiple frameworks and languages (e.g., TypeScript and Vue for the front-end, and Java and Spring for the back-end), and they do not need to transfer data between them.

The overall architecture of NASL and its supporting facilities is illustrated below:

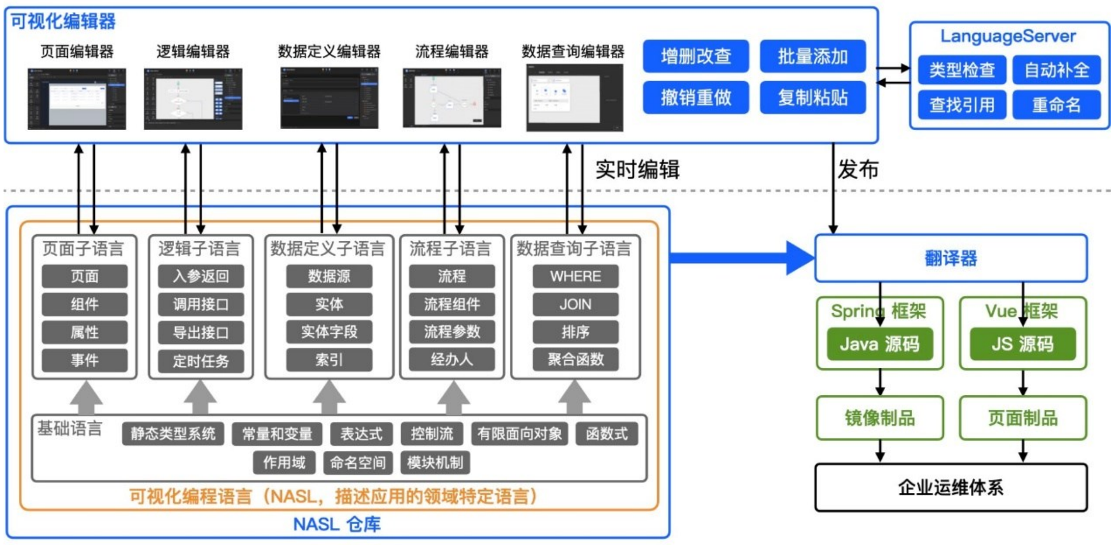


Fig. 3.14-2

During 2024, the majority of features are added to the IDE side. On the NASL side, features such as new textual syntax, exception handling, break and continue statements are added.

The following describes the base language, sub-languages, and supporting facilities.

Base language: The NASL base language incorporates language features common to object-oriented, functional, and other programming paradigms, and has the same expressive power as most general-purpose computer programming languages:

- A static type system that supports commonly used primitive, composite, collection, and meta data types.
- Union types and match expressions.
- Logic (function) definitions, which can use common control flows such as if, while, foreach and lambda expressions.
- Namespace, modules, and dependencies.provides a standard library of commonly used built-in functions.

NASL uses visualization to simplify complex language features, greatly reducing the learning threshold for users and matching the user profile of low-code groups.

Sub-languages: NASL sublanguages are DSLs that build on top of the base language by incorporating key features of traditional programming languages and frameworks for various subfields of Web applications:

- The Data Definition sublanguage, which is used to express concepts related to databases, tables, fields and indexes.
- Data query sublanguage, which is for expressing data query scenarios such as filtering, sorting, paging and aggregation.
- The page language, which is mainly used to express scenarios such as page layout, page interaction and page style.
- The process sublanguage, which is designed for expressing concepts related to the process domain such as manual tasks, automated tasks, exclusion gateways, etc.

The sub-languages are not independent of each other or cobbled together, but rather built on top of the base language and are more unified, explained as follows:

- Front-end, server-side, and entities all use a unified type definition.
- Front-end page logic, server-side logic, process logic can use unified expressions, statements, built-in functions standard library.
- Server-side logics called from front-end, interfaces called from logics, page jumping from process and other functions are all abstracted away from the underlying details, the user is senseless.

Complementary Facilities:

- Language Server: Includes type detection, type inference, jumping to definition, auto-completion and other capabilities to reduce the probability of programming errors and improve programming efficiency.
- Debugger: Includes breakpoint, step into, step over, resume, evaluate etc.
- Code Repository: Used for real-time storage of NASL code generated by users' building applications, and meets the characteristics of high-performance, high-availability, high-reliability and so on.
- Generator: NASL semantic compiler. Low-code platforms use Generator to compile NASL into general-purpose languages such as Java, JavaScript, etc., and then run NASL on the computer using the underlying general-purpose language runtime facilities such as the JVM.

- Upgrader: Used for compatibility issues that arise during the development of the NASL language.

Other aspects such as libraries and dependencies, compiler architecture, etc. are detailed in the Lightship low-code technology white paper:

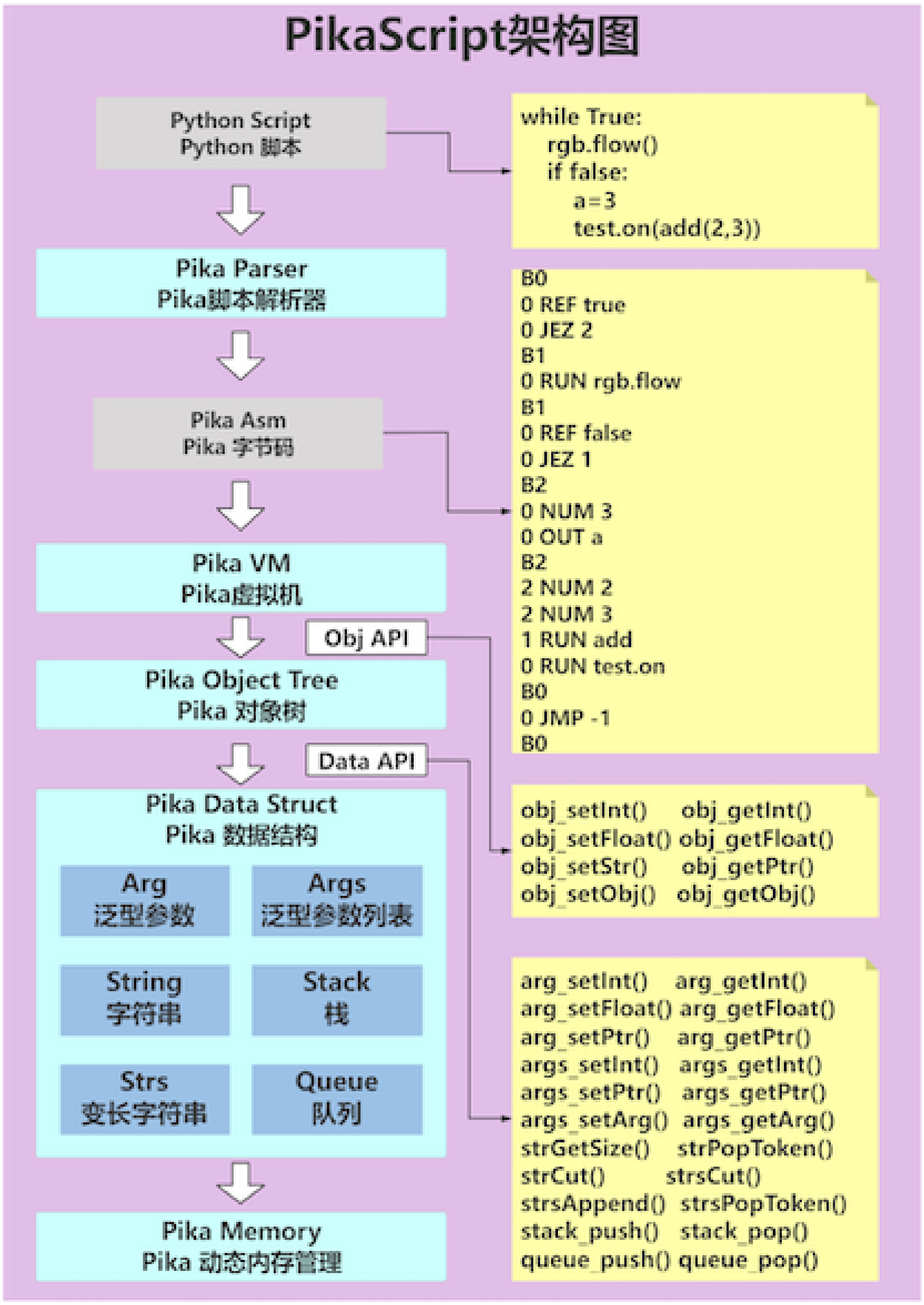
http://nasl.codewave.163.com/%E6%8A%80%E6%9C%AF%E7%99%BD%E7%9A%AE%E4%B9%A6V1.0-1118.pdf

3.15 PikaPython

Project Tags	Tool, Free, Opensource (MIT), General, Accepting Contribution
Tool Tags	Interpreter, Runtime Environment
Application Areas	Applied Computing
Homepage	https://pikapython.com
Repository	https://gitee.com/Lyon1998/pikapython

3.15.1 Introduction

PikaPython is a completely rewritten ultra-lightweight python engine. It has no zero dependencies and configuration. And it is extremely easy to deploy and extend , with a large number of Chinese documents and video materials . The project architecture is shown below:



```
while True:
    rgb.flow()
    if false:
        a=3
        test.on(add(2,3))
```

```
B0
0 REF true
0 JEZ 2
B1
0 RUN rgb.flow
B1
0 REF false
0 JEZ 1
B2
0 NUM 3
0 OUT a
B2
2 NUM 2
2 NUM 3
1 RUN add
0 RUN test.on
B0
0 JMP -1
B0
```

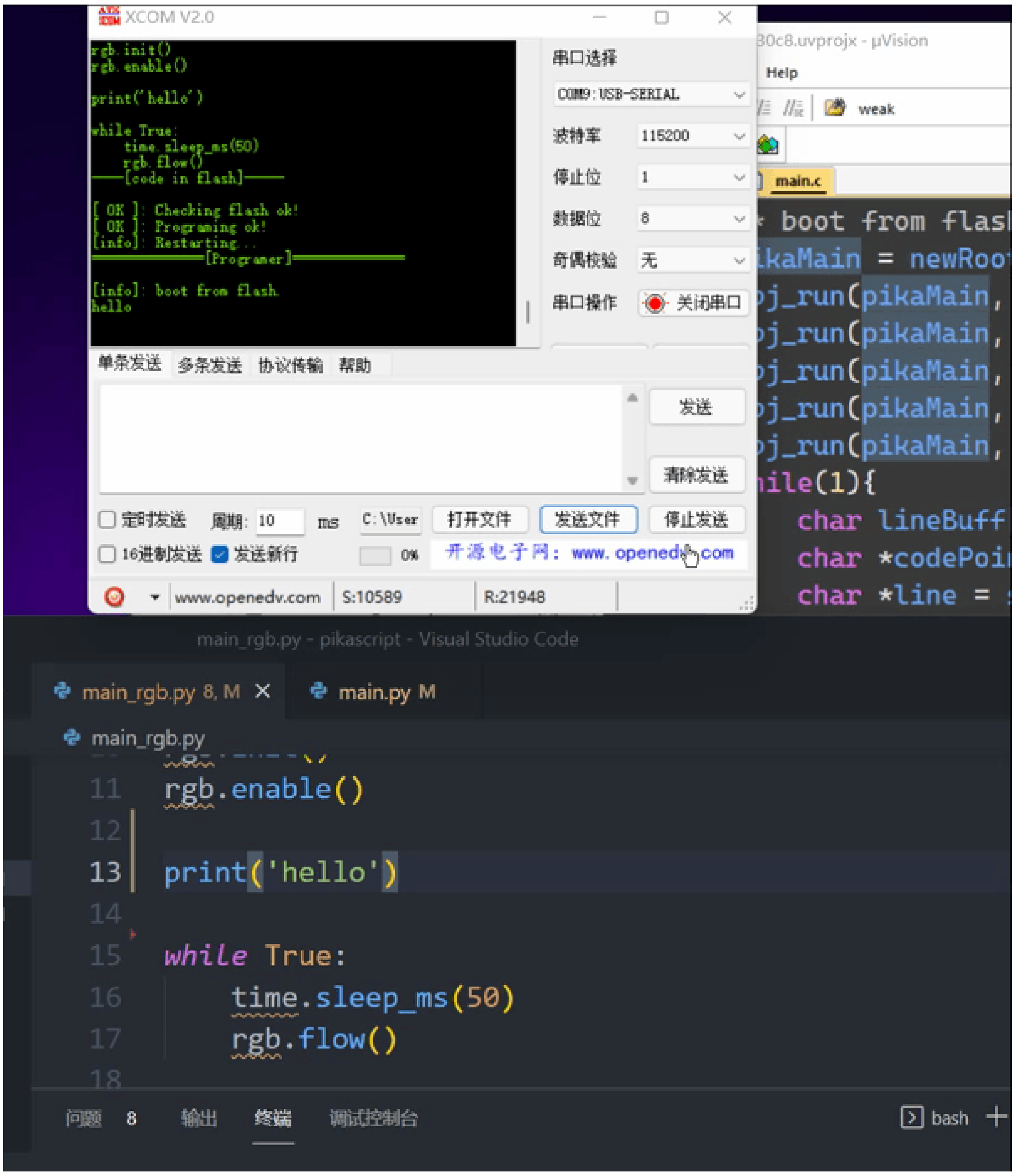
```
obj_setInt()   obj_getInt()
obj_setFloat() obj_getFloat()
obj_setStr()   obj_getPtr()
obj_setObj()   obj_getObj()
```

```
arg_setInt()    arg_getInt()
arg_setFloat()  arg_getFloat()
arg_setPtr()    arg_getPtr()
args_setInt()   args_getInt()
args_setPtr()   args_getPtr()
args_setArg()   args_getArg()
strGetSize()    strPopToken()
strCut()        strsCut()
strsAppend()    strsPopToken()
stack_push()    stack_pop()
queue_push()    queue_pop()
```

Fig. 3.15-1

- Running Environment: Support running on naked machines which configures are **RAM ⩾ 4kB** and **FLASH ⩾ 64kB**, such as stm32g030, stm32f103c8t6, esp8266 and so on;
- Development Environment: Support Keil, IAR, rt-thread studio, segger embedded

studio IDE; Support CMake, makeFile, Scons and other build tools; No dependency, no configuration, out-of-the-box and easy to integrate into existing C projects; Easy to expand C native functions; Support Cross-platform and kernel development in linux environment; Support serial port to download Python scripts, such as the Figure below:



```
rgb.init()
rgb.enable()

print('hello')

while True:
    time.sleep_ms(50)
    rgb.flow()
----[code in flash]----

[ OK ]: Checking flash ok!
[ OK ]: Programing ok!
[info]: Restarting...
===========[Programer]===========

[info]: boot from flash.
hello
```

```
* boot from flas
ikaMain = newRoo
oj_run(pikaMain,
oj_run(pikaMain,
oj_run(pikaMain,
oj_run(pikaMain,
oj_run(pikaMain,
hile(1){
   char lineBuff
   char *codePoi
   char *line = 
```

```python
rgb.enable()

print('hello')

while True:
    time.sleep_ms(50)
    rgb.flow()
```

Fig. 3.15-2

- Syntax Features: A subset of the python3 standard syntax; Supports python class and method definitions at compile time, Full support for encapsulation, inheritance, polymorphism, and module functionality based on the Pika pre-compiler; Supports python method calls, variable definitions, object construction, object release, and control flow (if\while) at runtime based on the Pika runtime kernel;

- Source Code Specification: Focus on source code readability, naming conventions, uniform standards, no macros, no global variables and full unit test based on googletest.

3.16 Qing

Project Tags	Language, Free, Opensource (MulanPSL V2.0), General, Accepting Contribution
Language Tags	General-Purpose Programming Language
Tool Tags	Interpreter, IDE
Application Areas	General
Homepage	https://qingyuyan.cn/
Repository	https://gitee.com/NjinN/Qing

3.16.1 Introduction

Qing is a programming language built entirely on Chinese language idioms and is aimed at teenagers, children and non-professionals.

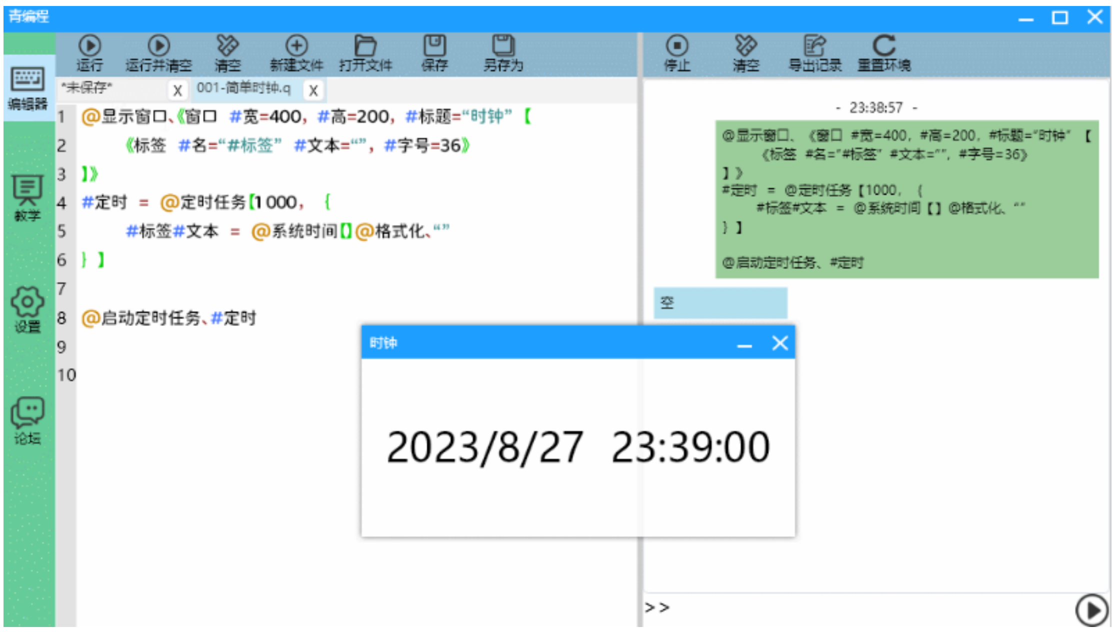
The main design components are as follows:

- The language core references the Lisp language. Lisp is known as a language for implementing programming languages, and its minimalist language core is very easy to implement. This makes the core language implementation of Cyan very simple, and makes it easy for opensource developers to participate in and contribute to the development of the language core.
- Syntactically references the JavaScript language. The JS programming language syntax is very simple, and its original design is also Lisp core, so it is very easy to implement. For users, there are fewer concepts to master and can be easily learnt and used.
- It is developed in C# and runs on the .Net platform can be said to be one of the most open, cross-platform compatibility of the best programming language, and itself has a good language ecology can be borrowed. Net platform can make the green language with good cross-platform compatibility, and at the same time can easily extend its functionality.
- currently uses dynamic link library DLL's to extend functionality. Cyan provides a simple way to encapsulate C# native functionality. By referring to sample projects, developers

can encapsulate the required functionality into a single DLL file, which can be easily shared and used.

Qing provides Interpreter, Editor, Android APP, while supporting Windows, Linux, OSX compatible, support GUI graphical interface programming.

Simple clock example:



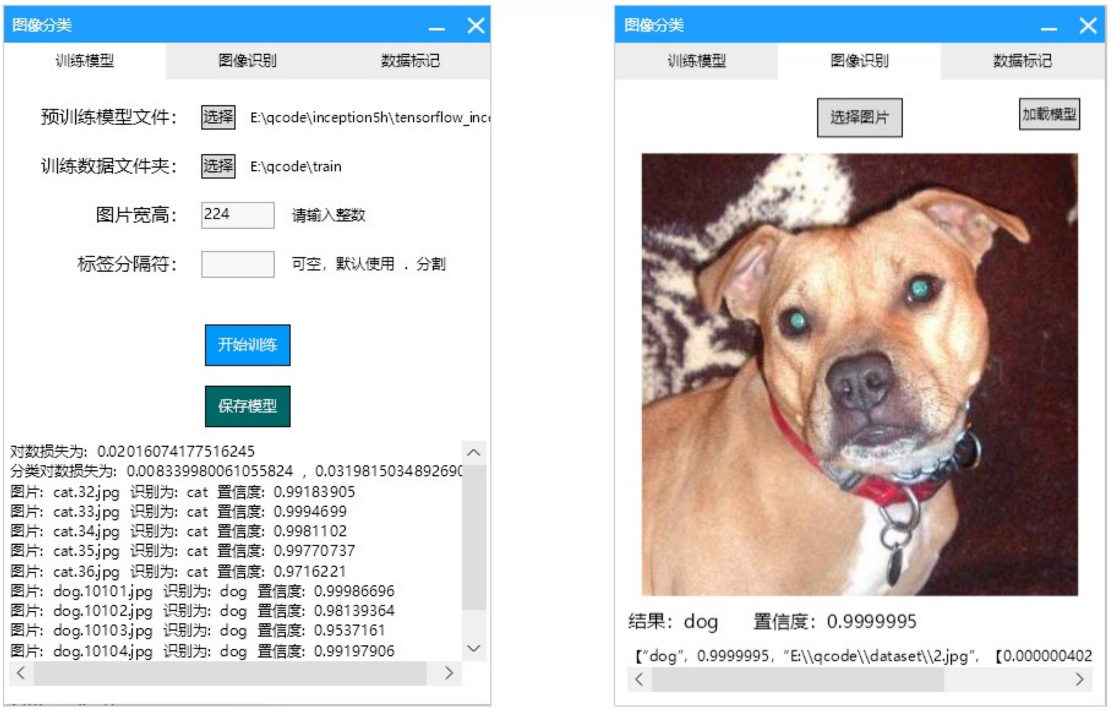
```
@显示窗口、《窗口 #宽=400，#高=200，#标题="时钟" 【
    《标签 #名="#标签" #文本=""，#字号=36》
】》
#定时 = @定时任务【1000， {
    #标签#文本 = @系统时间【】@格式化、""
} 】

@启动定时任务、#定时
```

```
- 23:38:57 -
@显示窗口、《窗口 #宽=400，#高=200，#标题="时钟" 【
    《标签 #名="#标签" #文本=""，#字号=36》
】》
#定时 = @定时任务【1000， {
    #标签#文本 = @系统时间【】@格式化、""
} 】

@启动定时任务、#定时
空
>>
```

2023/8/27 23:39:00

Fig. 3.16-1

AI Picture Classification:



预训练模型文件：E:\qcode\inception5h\tensorflow_inc

训练数据文件夹：E:\qcode\train

图片宽高：224 请输入整数

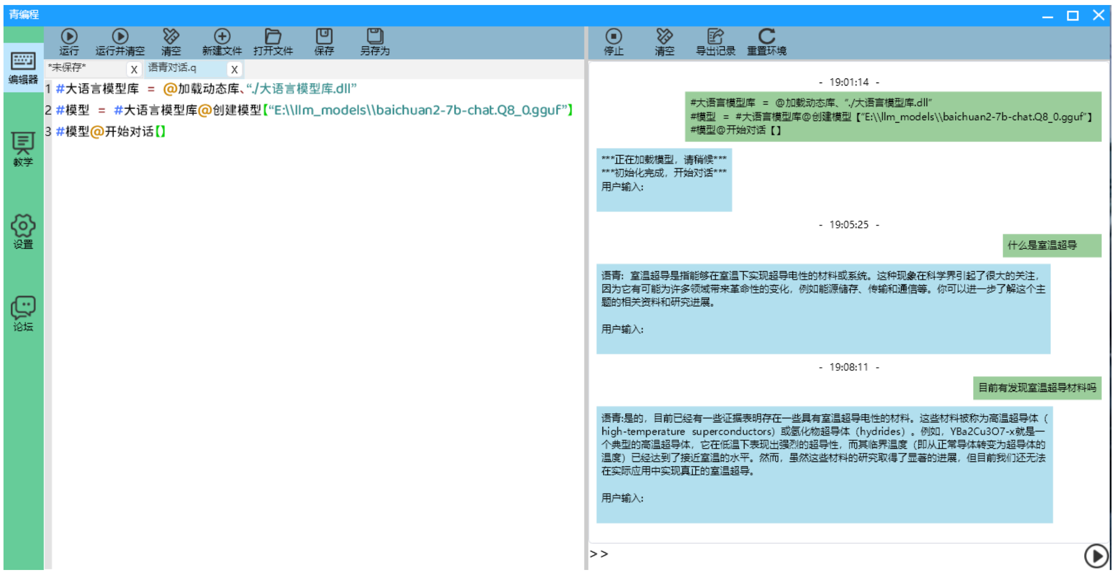
标签分隔符： 可空，默认使用 . 分割

```
对数损失为: 0.02016074177516245
分类对数损失为: 0.008339980061055824 , 0.0319815034892690
图片: cat.32.jpg 识别为: cat 置信度: 0.99183905
图片: cat.33.jpg 识别为: cat 置信度: 0.9994699
图片: cat.34.jpg 识别为: cat 置信度: 0.9981102
图片: cat.35.jpg 识别为: cat 置信度: 0.99770737
图片: cat.36.jpg 识别为: cat 置信度: 0.9716221
图片: dog.10101.jpg 识别为: dog 置信度: 0.99986696
图片: dog.10102.jpg 识别为: dog 置信度: 0.98139364
图片: dog.10103.jpg 识别为: dog 置信度: 0.9537161
图片: dog.10104.jpg 识别为: dog 置信度: 0.99197906
```

```
结果: dog    置信度: 0.9999995
["dog", 0.9999995, "E:\\qcode\\dataset\\2.jpg", [0.000000402
```

Fig. 3.16-2

Running Large Language Model:



```
#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【】
```

```
- 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:

>>
```

Fig. 3.16-3

3.17 RoarLang

Project Tags	Language, Free, Opensource(Apache2.0),General, Accepting Contribution
Language Tags	General, Concurrent Programming Language, Object Oriented Language
Tool Tags	General, Interpreter, Source Code Generation, Runtime Environment
Application Areas	General
Repository	https://gitee.com/openblock/openblock

3.17.1 Introduction

RoarLang is an open source, state-machine oriented, graphical, cross-platform, IDE-all-in-one scripting programming language. RoarLang is a fully business-oriented programming language and used to build a variety of business systems. It is not responsible for the implementation of the underlying technology, only responsible for the description of the business logic. IDE natively provides full-scenario capabilities, you can connect the entire line of business in a project.

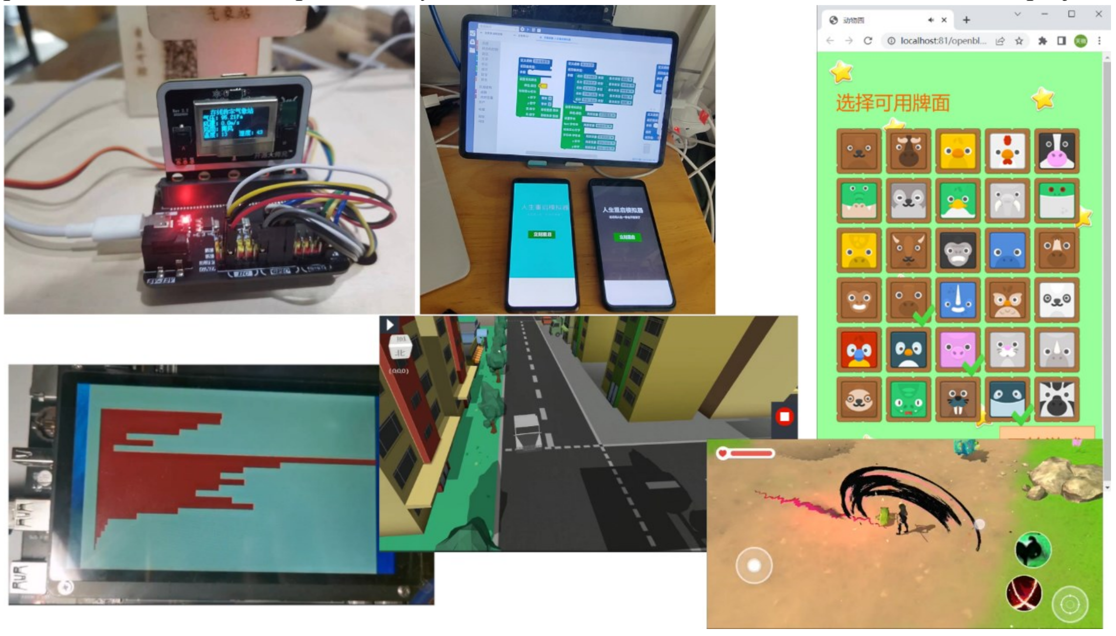


Fig. 3.17-1

RoarLang's compiler, interpreter, runtime and IDE are all opensource and freely modified. RoarLang IDE integrates resource management, static data management and other functions, you can secondary development to increase the specific field of the system. IDE provides graphical feedback capabilities and can be convenient for business people to understand the business logic. Static data can be edited by Excel and compiled into binary files along with the code during compilation, which reduces storage occupancy and improves operational efficiency. With the graphical internationalization capability, RoarLang can apply internationalization to all built-in and local libraries after the code is completed.

RoarLang has been experimentally commercialized in foreign enterprises, party politics, scientific research, education and other fields. Meanwhile, it has been widely used in the field of youth programming education, supporting two sessions of the Ministry of Education's National Primary and Secondary School Whitelist Competition.

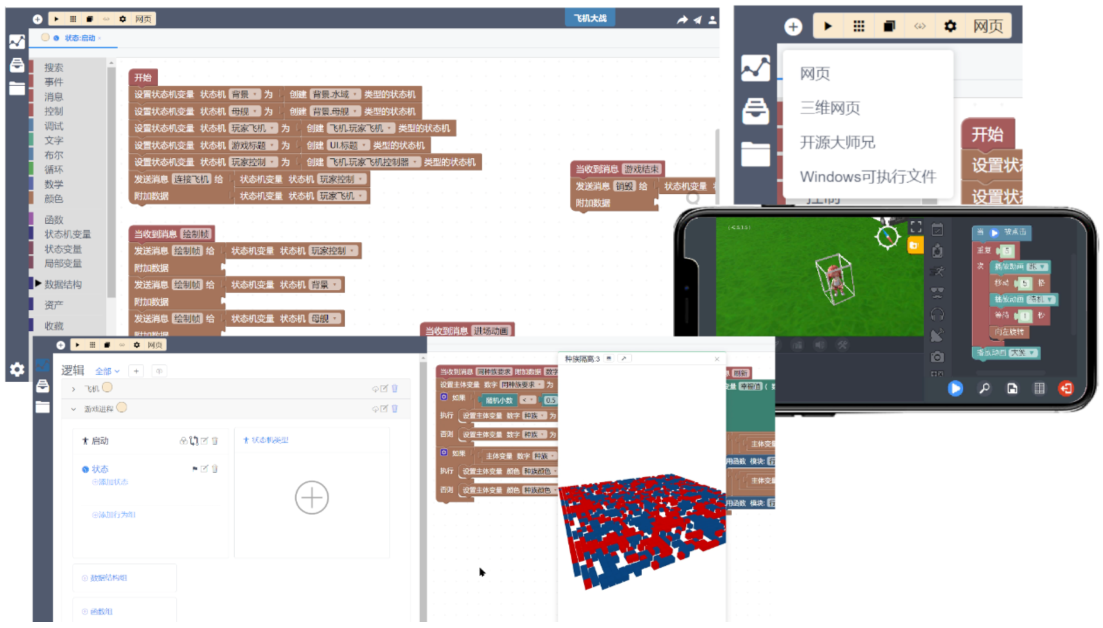


Fig. 3.17-2

Compilation and connectivity is written entirely in JavaScript and can be run in a browser or in Nodejs. The IDE is pure Html5 architecture and has no server-side requirements, any HTTP/HTTPS server can run it. Users on the business side do not need to configure the environment or install any software. The IDE is pure Html5 architecture and can be run on any HTTP/HTTPS server.

RoarLang is a state-machine oriented programming language, different states can listen to different events , execute different business logic , it can be very convenient to build a variety of complex business systems . State machines communicate with each other through asynchronous messages, you can make full use of multi-threaded, multi-process, distributed and other parallel technologies to achieve high concurrency and high performance.

RoarLang is a strongly typed language with support for custom data structures, and the native library can support generics. Functions are divided into three binding relationships: function, state machine behavior, and state behavior. Variables are divided into three scopes: state machine, state, and local.

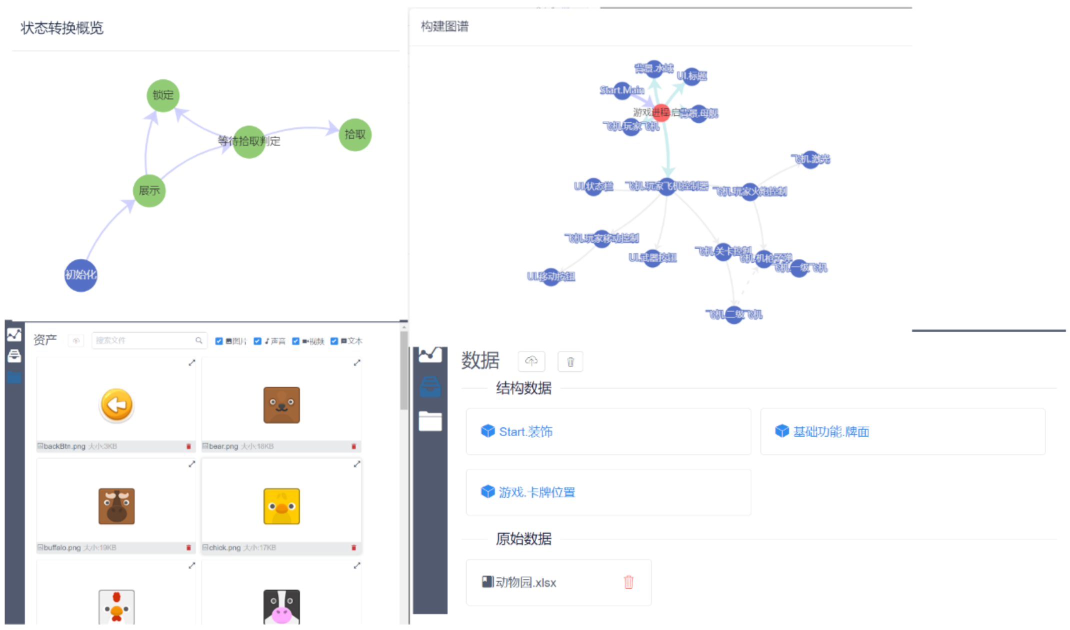


Fig. 3.17-3

Through the RoarLang VM to interpret the bytecode execution, or bytecode compiled into other languages, to achieve cross-platform and high-performance operation. At present, we mainly use VM.mjs to interpret bytecode in JS. Meanwhile, to C.mjs provides the function of compiling bytecode to C code. As RoarLang uses a lot of asynchronous logic, in can control concurrency and asynchrony at the technical layer according to the use of scenarios and technology stacks to give full play to the performance of the hardware, and can also support embedded, server, client, web page, executables and other operating environments, and can develop applications such as VR, WeChat applets, Internet of Things and so on. It meets the requirements of Information Technology Application Innovation and supports Harmony OS.

RoarLang means a doll in the shape of a lion that are suitable to keep on hand to motivate you.

3.18 TuLang

Project Tags	Language, Free, Opensource(AGPL-3.0), General, Accepting Contribution
Language Tags	General, Imperative Language
Tool Tags	General Compilation Tool
Application Areas	General, Industry Application
Repository	https://github.com/tu-lang/tu

3.18.1 Introduction

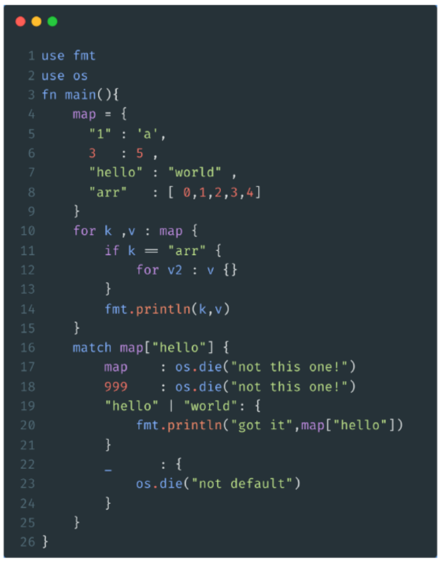
TuLang is a dynamically compiled programming language designed for general-purpose scenarios, the project was launched in 2018 and open-sourced in 2022, and has successfully implemented bootstrapping and is in the trial optimization phase

The language was originally designed out of a realization that existing programming languages were overly extreme in their pursuit of high performance and security, such as Rust and C++. Although these languages excel in performance and security, they place a considerable burden on developers in real-world program development, causing some fatigue.

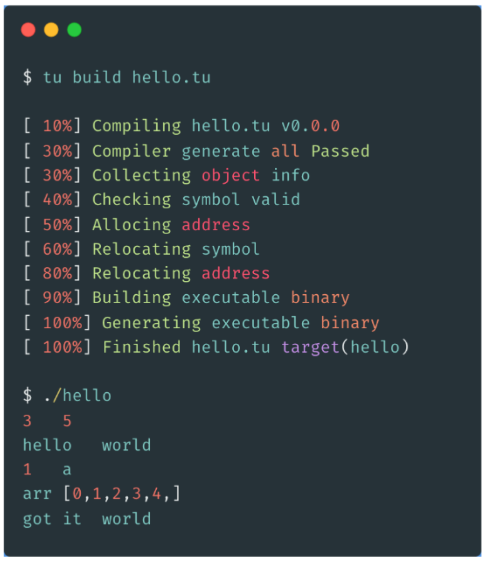
In contrast, current dynamic languages such as PHP, Python, and JavaScript have high development efficiency, but their performance is usually poor and they are mainly interpreted, and their extensibility is somewhat limited. In order to solve these problems, some underlying features have to be implemented by writing extension libraries for the C language.

The development goal of TuLang is to strike a balance between development efficiency, performance, and simplicity. In terms of development efficiency, it mainly adopts dynamic syntax, eliminating the need for cumbersome type annotations and allowing developers to focus on the implementation of business logic. In terms of performance, we write high-performance libraries with static syntax to provide a viable solution for high-performance scenarios, and provide mainstream rich and safe features such as stack coprocessing and multi-threaded GC. simplicity is another pursuit of the convex language, which is 100% zero-dependency and self-sufficient, with full-link bootstrap (compilation, assembly, and linking) and no need to rely on external toolchain support. Any amd64 Linux architecture.

Example of dynamic syntax:



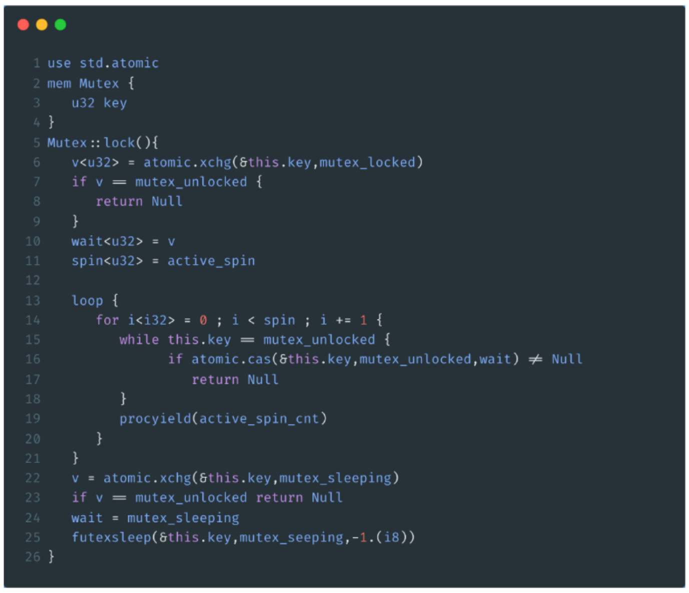
```
use fmt
use os
fn main(){
    map = {
      "1" : 'a',
      3   : 5 ,
      "hello" : "world" ,
      "arr"   : [ 0,1,2,3,4]
    }
    for k ,v : map {
        if k == "arr" {
            for v2 : v {}
        }
        fmt.println(k,v)
    }
    match map["hello"] {
        map     : os.die("not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

Fig. 3.18-1

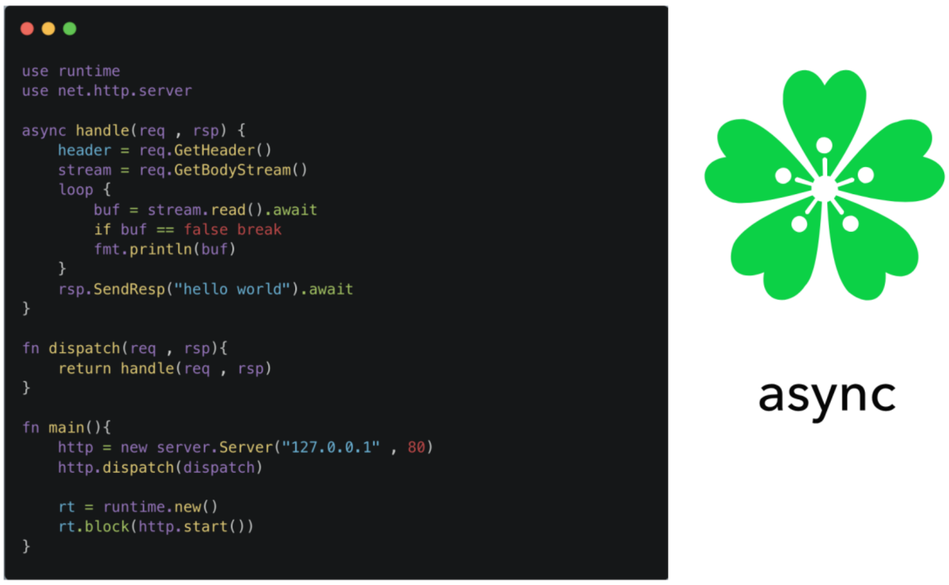
Example of static syntax.



```
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))
}
```

Fig. 3.18-2

Examples of feature syntax:



```
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())
}
```

Fig. 3.18-3

In the coming time, the development team will focus on the following directions:

- Improving the Multi-threaded Future Concurrent Management Framework;
- Optimizing Multithreaded GC Performance and Stability for Runtime;
- Enrichment package dependency management tools, documentation manuals;
- Enterprise real-world application projects;
- Designing Business Frameworks That Work.

TuLang welcomes programming language enthusiasts to build, polish, and work together to develop a language that belongs to us, the developers, ourselves.

3.19 Wa

Project Tags	Language, Free, Opensource(AGPL-3.0), General, Accepting Contribution
Language Tags	General, Imperative Language
Tool Tags	General
Application Areas	General, Computer Graphics, Modeling and Simulation
Homepage	https://wa-lang.org/
Repository	https://github.com/wa-lang/wa, https://gitcode.com/wa-lang/wa

3.19.1 Introduction

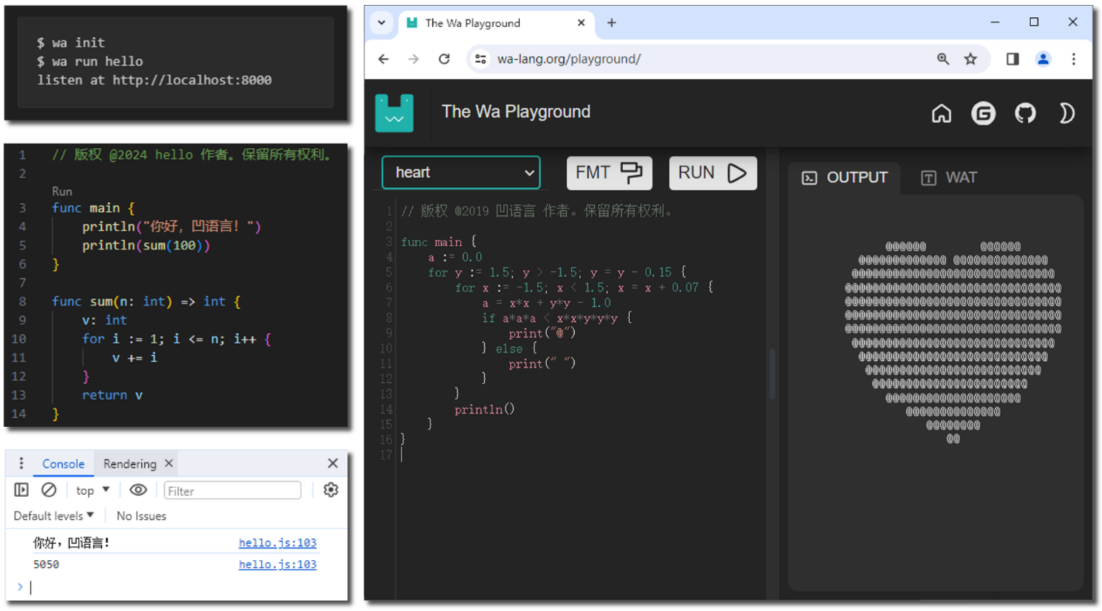
Wa (Chinese name "凹", which pronounced "Wa") is a programming language designed for WebAssembly (abbreviated Wasm). In terms of pronunciation, "Wa" is the first syllable of Wasm, in terms of shape, "凹" resembles the Wasm icon (a square with an opening missing at the top). The name of the Wa is inspired by the coincidence of this unique hieroglyphic character's double similarity in shape and pronunciation.

Wa is an opensource project maintained by Wuhan WaYuYan Technology Co., Ltd., its goal is to provide a clean, reliable, easy-to-use, strongly typed, compiled general-purpose programming language for high-performance web applications, Wa is currently in the engineering trial phase. The important milestones of it are as follows:

- In 2019, the project was launched;
- In July 2022, officially open sourced;
- In August 2023, the minimum viable version (MVP) was released;
- In November 2024, all syntactic features were implemented.

Easy-to-use is our design focus - the use of automatic memory management, strings as basic types, etc., all reflect this point. Wa compiler is a single-file executable program with built-in scaffolding that allows you to create a Wasm program in just three steps. In addition, Wa provides an online playground at https://wa-lang.org/playground, where you can write, compile, run and test

Wa code in the web page:



```
$ wa init
$ wa run hello
listen at http://localhost:8000
```

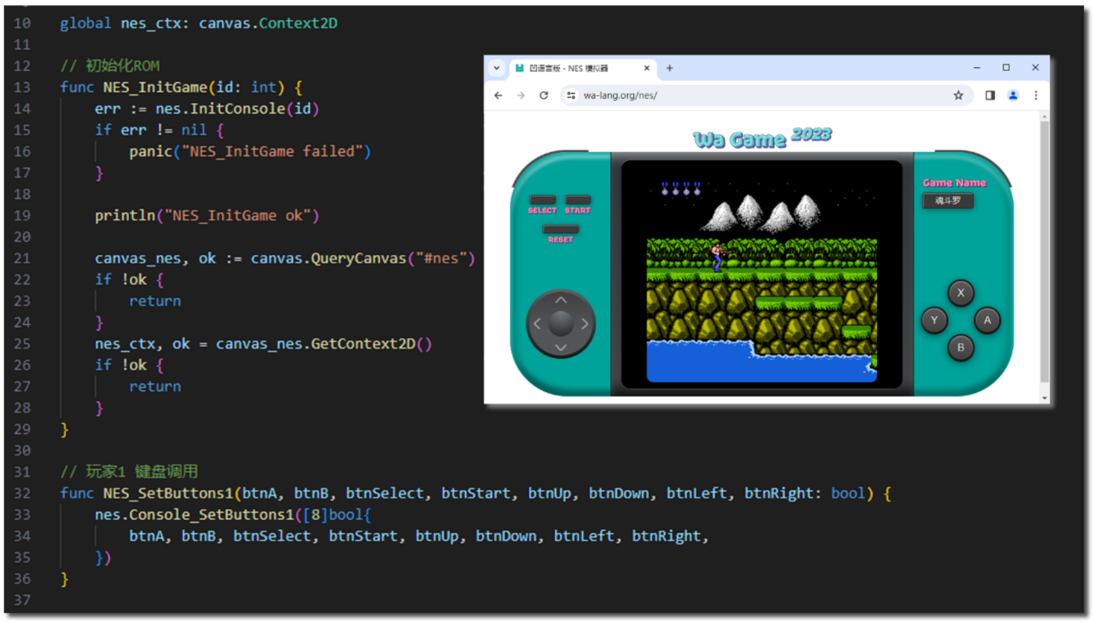
```
// 版权 @2024 hello 作者。保留所有权利。

Run
func main {
    println("你好，凹语言！")
    println(sum(100))
}

func sum(n: int) => int {
    v: int
    for i := 1; i <= n; i++ {
        v += i
    }
    return v
}
```

```
你好，凹语言!                hello.js:103
5050                         hello.js:103
```

```
// 版权 @2019 凹语言 作者。保留所有权利。

func main {
    a := 0.0
    for y := 1.5; y > -1.5; y = y - 0.15 {
        for x := -1.5; x < 1.5; x = x + 0.07 {
            a = x*x + y*y - 1.0
            if a*a*a < x*x*y*y*y {
                print("@")
            } else {
                print(" ")
            }
        }
        println()
    }
}
```

Fig. 3.19-1

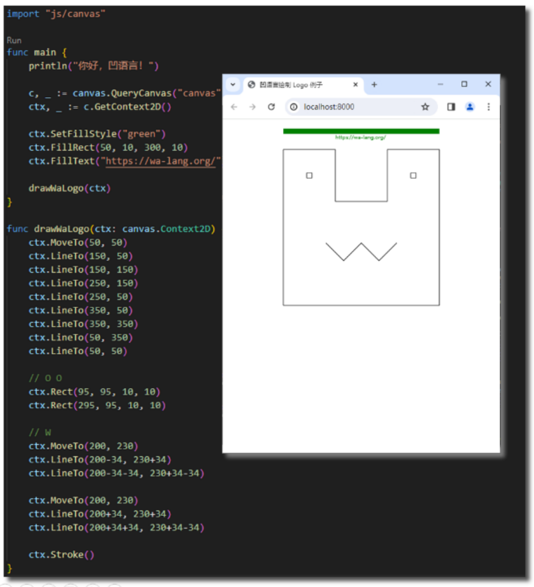
Although Wa is still in its early stages, it has already shown strong performance. The Nintendo FC emulator developed with it: https://wa-lang.org/nes can run various FC-ROMs smoothly (in comparison, using the same emulation method, the performance of the FC emulator developed by Python is only 1% of the real machine):



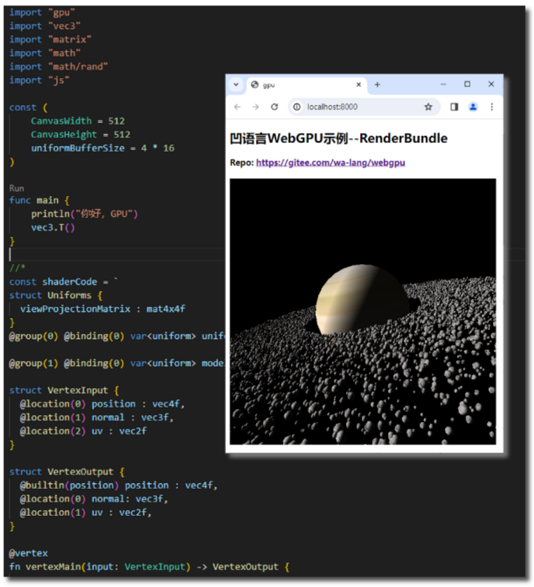
```
global nes_ctx: canvas.Context2D

// 初始化ROM
func NES_InitGame(id: int) {
    err := nes.InitConsole(id)
    if err != nil {
        panic("NES_InitGame failed")
    }

    println("NES_InitGame ok")

    canvas_nes, ok := canvas.QueryCanvas("#nes")
    if !ok {
        return
    }
    nes_ctx, ok = canvas_nes.GetContext2D()
    if !ok {
        return
    }
}

// 玩家1 键盘调用
func NES_SetButtons1(btnA, btnB, btnSelect, btnStart, btnUp, btnDown, btnLeft, btnRight: bool) {
    nes.Console_SetButtons1([8]bool{
        btnA, btnB, btnSelect, btnStart, btnUp, btnDown, btnLeft, btnRight,
    })
}
```

Fig. 3.19-2

Application areas of Wa include XR, games, industrial design, geospatial information system, and other computationally intensive web applications. The project team is developing graphics and image support libraries for these applications:



```
import "js/canvas"

Run
func main {
    println("你好，凹语言！")

    c, _ := canvas.QueryCanvas("canvas"
    ctx, _ := c.GetContext2D()

    ctx.SetFillStyle("green")
    ctx.FillRect(50, 10, 300, 10)
    ctx.FillText("https://wa-lang.org/"

    drawWaLogo(ctx)
}

func drawWaLogo(ctx: canvas.Context2D)
    ctx.MoveTo(50, 50)
    ctx.LineTo(150, 50)
    ctx.LineTo(150, 150)
    ctx.LineTo(250, 150)
    ctx.LineTo(250, 50)
    ctx.LineTo(350, 50)
    ctx.LineTo(350, 350)
    ctx.LineTo(50, 350)
    ctx.LineTo(50, 50)

    // O O
    ctx.Rect(95, 95, 10, 10)
    ctx.Rect(295, 95, 10, 10)

    // W
    ctx.MoveTo(200, 230)
    ctx.LineTo(200-34, 230+34)
    ctx.LineTo(200-34-34, 230+34-34)

    ctx.MoveTo(200, 230)
    ctx.LineTo(200+34, 230+34)
    ctx.LineTo(200+34+34, 230+34-34)

    ctx.Stroke()
}
```



凹语言WebGPU示例--RenderBundle

Repo: https://gitee.com/wa-lang/webgpu

```
import "gpu"
import "vec3"
import "matrix"
import "math"
import "math/rand"
import "js"

const (
    CanvasWidth = 512
    CanvasHeight = 512
    uniformBufferSize = 4 * 16
)

Run
func main {
    println("你好，GPU")
    vec3.T()
}

//*
const shaderCode = `
struct Uniforms {
  viewProjectionMatrix : mat4x4f
}
@group(0) @binding(0) var<uniform> unif

@group(1) @binding(0) var<uniform> mode

struct VertexInput {
  @location(0) position : vec4f,
  @location(1) normal : vec3f,
  @location(2) uv : vec2f
}

struct VertexOutput {
  @builtin(position) position : vec4f,
  @location(0) normal: vec3f,
  @location(1) uv : vec2f,
}

@vertex
fn vertexMain(input: VertexInput) -> VertexOutput {
```

Fig. 3.19-3

From 2019 to 2023, the first five-year plan of the Wa project, the development team has basically achieved the goal of "usability"; for the second five-year plan, our goal is "easy to use". As the first year of the second five-year plan, the progress of the project in 2024 is mainly reflected in the following aspects:

- Added function overloading, operator overloading, embed, map, defer, and complex number support to realize all syntax features;
- Developed a built-in wat to wasm module to realize the full self-development of the compiler backend;
- Provided support for frameworks such as p5, wasm4, and Arduino Nano 33;
- Fixed a lot of bugs.

In the coming time, we will focus on the following directions:

- Functional base library and framework development;
- Backend and runtime reconstruction, performance improvement;
- Practical project implementation;
- Tool chain, manual, and scaffolding optimization.

Wa is the result of community cooperation. The compiler is written in Golang and the standard library is written in Wa. Programming language enthusiasts are always welcome to gather around and build together!

3.20 XLang

Project Tags	Language, Free, Opensource(AGPL-3.0), General, Accepting Contribution
Language Tags	General (Scripting Language), Imperative Language, Functional Language , Extensible Language
Tool Tags	Interpreter, Source Code Generation
Application Areas	General, Industry Applications
Homepage	https://nop-platform.github.io/projects/nop-entropy/docs/dev-guide/xlang/
Repository	https://github.com/entropy-cloud/nop-entropy

3.20.1 Introduction

XLang is a scripting language that combines XML tag syntax and JavaScript syntax. It was designed by canonical. The XLang language is one of the underlying technologies of the Nop low-code platform and is the world's first programming language with built-in support for Delta merge operators from reversible computation theory. XLang's hello world is as follow:

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

The Nop platform adopts the language-oriented programming paradigm (Language Oriented Programming). When developing applications, it does not directly use general-purpose programming languages (such as Java, C#) for development, but first defines a domain-specific language (DSL), and then uses the DSL to express business. In order to quickly develop and expand the DSL language, we need a meta-model language that can define the DSL. syntax structure, as well as a series of mechanisms to quickly implement the DSL interpreter, syntax-guided translation, etc. The XLang language XLang includes sub-languages such as the XDef meta-model definition language, the Xpl template language, the XScript expression language, and the XTransform structural transformation language. They together form a complete DSL development infrastructure. By adding a simple XDef meta-model definition, we can automatically obtain the corresponding DSL parser, validator, IDE plugin, debugging tool, etc., and automatically add general language features such as module decomposition, delta customization, and meta-programming to the DSL domain language.

- Defining a new DSL through XDef meta-model definition:

```
<!--
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>
```

- **Interweaving of XML and Expression Syntax.** XLang does not use JSX syntax to implement XML-like syntax, but continues to use XML syntax and extends the Template expression syntax in JavaScript:

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

Equivalent to

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

The overall structure of XLang strictly complies with the XML format requirements, so it meets the homoiconicity principle first introduced by Lisp language when used as a template language to generate XML, making it particularly suitable for meta-programming and macro function implementation.

XLang modifies the parsing format of the Template expression syntax in JavaScript, recognizing the content between backtick characters as a string to be parsed at compile time, rather than an Expression list. This allows XLang to use this syntax form to extend support for more DSL formats, such as introducing a LinQ syntax similar to C#.

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

Implementing a parser similar to LinQ syntax is very simple, just define a static function in Java, marked with @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);

    }
```

- Macro Functions and Compile-Time Execution. XLang supports compile-time meta-programming, allowing Turing-complete code to be executed at compile time and dynamically generating new syntax structures to be compiled.

```
 <!--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's Delta Generation and Merge Mechanism. All DSLs in the Nop platform support the x-extends delta merge mechanism, which implements the computational pattern required by reversible computation theory:

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

Specifically, all DSLs support x:gen-extends and x:post-extends configuration sections, which are Generators executed at compile time, using the XPL template language to dynamically generate model nodes, allowing multiple nodes to be generated at once, and then merged in order, with the specific merge order defined as follows:

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

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

The merge result is:

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

The current model will overwrite the results of x:gen-extends and x:extends, while x:post-extends will overwrite the current model.

With the help of x:extends and x:gen-extends, We can effectively implement the decomposition and combination of DSL. For more details, see https://zhuanlan.zhihu.com/p/612512300.

- Extensible Syntax. Similar to Lisp language, the syntax of XLang can be extended through macro functions and tag functions. New syntax nodes can be introduced through <c:lib>, and then structural transformations can be implemented within the node through macro functions and other mechanisms.

```
<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> introduces a DSL for validation. The Validator tag will use the macro function mechanism to parse the node content at compile time and translate it into an XLang Expression for execution.

Charter 4 About us

PLOC:

The Programming Language Open Community (PLOC) is a professional community of programming languages and compilers spontaneously formed by Chinese practitioners. The community is based on the following realities:

- Programming languages and compilers are the real "root technologies" and "industrial mother machines" of the software industry, but our country has made little progress in this industry;
- Decision-makers and the industry are beginning to realize the importance of PL technology;
- There is no regular pattern in the development experience of the widely used programming languages today;

There are many programming language projects in China, but practitioners are highly dispersed, no one can give a representative and comprehensive panoramic view, and there is also a lack of industry voice channels.

Unlike other loose SIGs, PLOC has a clear program and vision, a complete system charter, and a strict organizational structure. It has a permanent decision-making body (CPLOC) and a comprehensive service agency (CPLOC Secretariat), and promotes community construction and activities through professional committees.

"Practitioners supporting each other" is the core concept of PLOC. Programming language projects have a high threshold for startup, high commercialization difficulty, and the number of practitioners is relatively small, so communication and mutual support in the industry is especially important. We hope that through the establishment of this community, we can promote the startup of more programming language projects (increase the base number), prolong the survival time of the projects (increase the success factor), and help the development of rooted software in China.

HBSIA:

Hubei Software Industry Association (HBSIA), established in 2000, is a 5A-level social organization engaged in software research and development as well as information services.

Membership Composition: The association currently has approximately 1,200 official member units, all of which are economic organizations engaged in software and information service research and development, sales, scientific research, and related fields. The current chairman (legal representative) is Zeng Jun, the chairman of Fiberhome Telecommunication Technologies Co., Ltd.

Social Recognition: In 2010, it was awarded the title of "National Advanced Social Organization" by the Ministry of Civil Affairs. In 2017, it received the "National Youth Civilization Unit" title from the Central Committee of the Communist Youth League of China. In 2012 and 2024, it was rated as a "5A-level Social Organization" by the Hubei Provincial Department of Civil Affairs.

Service Brand: Over the years, HBSIA has aimed to "strive to be a leading flag in social organization services, build a first-class national software industry association, and serve the high-quality development of the software industry." Adhering to professionalization, standardization, and marketization, with the "Scenario Road Plan" as its long-term vision, the association has established a service system of "2 Conferences and 10 Platforms," including the Pomegranate Conference, specialized committees, and platforms for technology supply chains, markets, talents, capital, and software and information technology solutions. It serves administrative agencies, public institutions, research institutes, universities, and enterprises in fields related to the digital economy, totaling over 2,500 organizations, with service recipients spread across more than 20 provinces, municipalities directly under the central government, and autonomous regions nationwide. The association plays an active role in strengthening party building leadership, assisting government decision-making, promoting industry self-discipline, improving consultative democracy, cultivating digital talents, prioritizing employment, developing digital standards, facilitating exchange and cooperation, empowering enterprise development, and fulfilling public welfare responsibilities. It has been commended by leaders of the Ministry of Civil Affairs as a "Model for social organizations."

Appendix

Language Category:

a. General Programming Language	b. Parallel Programming Language	c. Concurrent Programming Language
d. Distributed Programming Language	e. Imperative Language	f. Object Oriented Language
g. Functional Language	h. Constraint and Logic Language	i. Dataflow Language
j. Extensible Language	k. Assembly Language	

Tool Category:

a. General Compiler	b. Interpreter	c. Incremental Compiler
d. Retargetable Compiler	e. Just-in-time Compiler	f. Dynamic Compiler
g. Translator Writing Systems and Compiler Generators	h. Source Code Generation	i. Runtime Environment
j. Preprocessor	k. Parser	

Application Area:

a. General Computation	b. Theory of Computation	c. Mathematics of Computing
d. Networks	e. Information System	f. Security
g. Machine Learning	h. Artificial Intelligence	i. Parallel Computing
j. Concurrent Computing	k. Distributed Computing	l. Modelling and Simulation
m. Computer Graphics	n. Applied Computing / Industry Applications	

https://cdn-static.gitcode.com/doc/CNPL-2024-EN.pdf

Programming Language Open Community (PLOC)
WeChat Official Account

Hubei Software Industry Association
WeChat Official Account