2025 年国家汇编执行手册

Version 1.0

2025 年 3 月

目录

第一章 国家汇编执行环境：概念与总览 7

1.1 什么是国家汇编？ 7

1.1.1 核心概念 7

1.1.2 与传统汇编的区别 8

1.1.3 适用场景 9

1.2 执行环境总览 10

1.2.1 整体架构 10

1.2.2 组件清单 11

1.2.3 版本兼容性 12

1.3 术语表 13

1.3.1 通用术语 13

1.3.2 缩略语 14

第二章 安装与配置 15

2.1 系统要求 15

2.1.1 硬件要求 15

2.1.2 操作系统支持 16

2.1.3 依赖项检查 17

2.2 安装步骤 18

2.2.1 下载安装包 18

2.2.2 安装流程 19

2.2.3 验证安装 20

2.3 配置文件详解 21

2.3.1 主配置文件 21

2.3.2 环境变量 22

2.3.3 日志配置 23

2.4 卸载与升级 24

2.4.1 卸载流程 24

2.4.2 升级流程 25

第三章 基础语法与数据类型 26

3.1 词法规则 26

3.1.1 标识符 26

3.1.2 关键字 27

3.1.3 注释 28

3.2 数据类型 29

3.2.1 基本类型 29

3.2.2 复合类型 30

3.2.3 类型转换 31

3.3 变量与常量 32

3.3.1 变量声明 32

3.3.2 常量定义 33

3.3.3 作用域规则 34

3.4 运算符 35

3.4.1 算术运算符 35

3.4.2 比较运算符 36

3.4.3 逻辑运算符 37

3.4.4 位运算符 38

第四章 控制流与函数 39

4.1 条件语句 39

4.1.1 if 语句 39

4.1.2 switch 语句 40

4.2 循环语句 41

4.2.1 for 循环 41

4.2.2 while 循环 42

4.2.3 循环控制 43

4.3 函数定义与调用 44

4.3.1 函数声明 44

4.3.2 参数传递 45

4.3.3 返回值 46

4.4 递归 47

4.4.1 递归原理 47

4.4.2 尾递归优化 48

第五章 内存管理 49

5.1 内存模型 49

5.1.1 内存布局 49

5.1.2 栈与堆 50

5.2 指针 51

5.2.1 指针基础 51

5.2.2 指针运算 52

5.2.3 空指针与野指针 53

5.3 动态内存分配 54

5.3.1 malloc 与 free 54

5.3.2 内存泄漏检测 55

5.4 智能指针 56

5.4.1 unique_ptr 56

5.4.2 shared_ptr 57

第六章 模块与包管理 58

6.1 模块系统 58

6.1.1 模块定义 58

6.1.2 模块导入 59

6.2 包管理器 60

6.2.1 包管理器安装 60

6.2.2 常用命令 61

6.3 依赖管理 62

6.3.1 依赖声明 62

6.3.2 版本约束 63

6.3.3 锁文件 64

第七章 并发与异步 65

7.1 线程 65

7.1.1 线程创建 65

7.1.2 线程同步 66

7.2 互斥锁与条件变量 67

7.2.1 互斥锁 67

7.2.2 条件变量 68

7.3 异步编程 69

7.3.1 async 与 await 69

7.3.2 Future 与 Promise 70

第八章 网络编程 71

8.1 套接字 71

8.1.1 TCP 套接字 71

8.1.2 UDP 套接字 72

8.2 HTTP 客户端 73

8.2.1 发送请求 73

8.2.2 处理响应 74

8.3 服务器开发 75

8.3.1 简单服务器 75

8.3.2 路由与中间件 76

第九章 测试与调试 77

9.1 单元测试 77

9.1.1 测试框架 77

9.1.2 断言 78

9.2 调试技巧 79

9.2.1 调试器使用 79

9.2.2 日志调试 80

9.3 性能分析 81

9.3.1 性能剖析 81

9.3.2 内存分析 82

第十章 部署与运维 83

10.1 构建发布版本 83

10.1.1 构建配置 83

10.1.2 交叉编译 84

10.2 容器化部署 85

10.2.1 Docker 镜像 85

10.2.2 编排与调度 86

10.3 监控与告警 87

10.3.1 指标采集 87

10.3.2 告警规则 88

附录 89

A 错误码速查表 89

B 命令行参考 90

C 配置项索引 91

D 常见问题 FAQ 92

第一章 国家汇编执行环境：概念与总览

1.1 什么是国家汇编？

1.1.1 核心概念

国家汇编（National Assembly）是一种面向国家级基础设施的低级编程语言，其设计目标是在保证极高执行效率的同时，提供对硬件资源的精确控制。

1.1.2 与传统汇编的区别

特性	传统汇编	国家汇编
可移植性	低，绑定单一架构	中，支持同族架构
内存安全	无保障	编译期检查
调试支持	基础	完整符号与源码映射
学习曲线	陡峭	较为平缓

1.1.3 适用场景

- 操作系统内核与引导程序
- 嵌入式实时控制系统
- 高频交易与低延迟网络栈
- 密码学算法的恒定时间实现

1.2 执行环境总览

1.2.1 整体架构

执行环境由以下层次组成：

1. 硬件抽象层（HAL）
2. 运行时库（Runtime）
3. 用户程序

1.2.2 组件清单

组件	说明
`nasm`	汇编器
`nld`	链接器
`nrun`	运行时加载器
`ndbg`	调试器

1.2.3 版本兼容性

> 注意：本手册仅适用于国家汇编 1.x 系列，2.x 系列将引入不兼容的语法变更。

1.3 术语表

1.3.1 通用术语

- **ABI**：应用二进制接口
- **IR**：中间表示

1.3.2 缩略语

缩略语	全称
MMU	Memory Management Unit
TLB	Translation Lookaside Buffer

第二章 安装与配置

2.1 系统要求

2.1.1 硬件要求

- CPU：x86_64 架构，支持 SSE4.2
- 内存：最低 4 GB，推荐 8 GB 及以上
- 磁盘：至少 2 GB 可用空间

2.1.2 操作系统支持

操作系统	版本	状态
Linux	5.4 及以上	完全支持
macOS	12 及以上	完全支持
Windows	10 及以上	实验性支持

2.1.3 依赖项检查

```
$ nasm --version
$ nld --version
```

2.2 安装步骤

2.2.1 下载安装包

从官方镜像站下载对应平台的压缩包，并校验 SHA-256 摘要。

2.2.2 安装流程

1. 解压安装包到目标目录
2. 将 `bin/` 加入 `PATH`
3. 执行 `nasm --version` 验证

2.2.3 验证安装

```
$ nasm --version
nasm 1.0.3 (build 2025-01-12)
```

2.3 配置文件详解

2.3.1 主配置文件

主配置文件位于 `~/.nasm/config.toml`，格式为 TOML。

2.3.2 环境变量

变量	说明
`NASM_HOME`	安装根目录
`NASM_LOG`	日志级别

2.3.3 日志配置

```
[log]
level = "info"
file = "/var/log/nasm.log"
```

2.4 卸载与升级

2.4.1 卸载流程

删除安装目录与 `~/.nasm/` 即可完成卸载。

2.4.2 升级流程

> 升级前请备份 `config.toml`，避免配置被覆盖。

第三章 基础语法与数据类型

3.1 词法规则

3.1.1 标识符

标识符由字母、数字与下划线组成，不得以数字开头。

3.1.2 关键字

关键字	用途
`let`	变量绑定
`fn`	函数定义
`return`	返回值

3.1.3 注释

```
// 单行注释
/* 多行注释 */
```

3.2 数据类型

3.2.1 基本类型

- 整数：`i32`、`i64`、`u32`、`u64`
- 浮点：`f32`、`f64`
- 布尔：`bool`

3.2.2 复合类型

类型	说明
数组	固定长度同类元素
元组	异类元素组合

3.2.3 类型转换

```
let x: i32 = 42;
let y: f64 = x as f64;
```

3.3 变量与常量

3.3.1 变量声明

变量默认不可变，使用 `mut` 关键字声明可变变量。

3.3.2 常量定义

常量在编译期求值，命名通常采用全大写。

3.3.3 作用域规则

- 块级作用域
- 内层可遮蔽外层同名绑定

3.4 运算符

3.4.1 算术运算符

运算符	含义
`+`	加
`-`	减
`*`	乘
`/`	除

3.4.2 比较运算符

比较运算符返回 `bool` 值。

3.4.3 逻辑运算符

```
let a = true && false;
let b = true || false;
```

3.4.4 位运算符

位运算仅适用于整数类型。

第四章 控制流与函数

4.1 条件语句

4.1.1 if 语句

```
if x > 0 {
    // ...
}
```

4.1.2 switch 语句

switch 语句必须覆盖所有可能取值，否则编译报错。

4.2 循环语句

4.2.1 for 循环

for 循环用于遍历迭代器。

4.2.2 while 循环

while 循环在条件为真时持续执行。

4.2.3 循环控制

- `break`：跳出当前循环
- `continue`：进入下一次迭代

4.3 函数定义与调用

4.3.1 函数声明

```
fn add(a: i32, b: i32) -> i32 {
    return a + b;
}
```

4.3.2 参数传递

参数默认按值传递。

4.3.3 返回值

写法	说明
`return x;`	显式返回
末尾表达式	隐式返回

4.4 递归

4.4.1 递归原理

递归函数必须有明确的终止条件。

4.4.2 尾递归优化

> 尾递归可被编译器优化为循环，避免栈溢出。

第五章 内存管理

5.1 内存模型

5.1.1 内存布局

进程内存自低地址向高地址依次为：代码段、数据段、堆、栈。

5.1.2 栈与堆

区域	分配方式	生命周期
栈	自动	函数返回即释放
堆	手动	显式释放

5.2 指针

5.2.1 指针基础

```
let p: *i32 = &x;
let v = *p;
```

5.2.2 指针运算

指针运算以元素大小为步长。

5.2.3 空指针与野指针

- 空指针：`null`
- 野指针：指向已释放或未初始化内存

5.3 动态内存分配

5.3.1 malloc 与 free

```
let buf = malloc(1024);
free(buf);
```

5.3.2 内存泄漏检测

> 每次 `malloc` 必须有对应的 `free`，否则视为泄漏。

5.4 智能指针

5.4.1 unique_ptr

`unique_ptr` 独占所有权，不可复制，可移动。

5.4.2 shared_ptr

`shared_ptr` 通过引用计数共享所有权。

第六章 模块与包管理

6.1 模块系统

6.1.1 模块定义

```
mod math {
    pub fn add(a: i32, b: i32) -> i32 { a + b }
}
```

6.1.2 模块导入

使用 `use` 关键字导入模块或其成员。

6.2 包管理器

6.2.1 包管理器安装

包管理器随工具链一同安装，无需单独下载。

6.2.2 常用命令

命令	说明
`pkg init`	初始化项目
`pkg add`	添加依赖
`pkg build`	构建项目

6.3 依赖管理

6.3.1 依赖声明

```
[dependencies]
foo = "1.2.0"
```

6.3.2 版本约束

写法	含义
`1.2.0`	精确版本
`^1.2.0`	兼容版本

6.3.3 锁文件

锁文件记录解析后的精确版本，应提交到版本控制。

第七章 并发与异步

7.1 线程

7.1.1 线程创建

```
let h = thread::spawn(|| {
    // ...
});
h.join();
```

7.1.2 线程同步

线程同步常用互斥锁与条件变量。

7.2 互斥锁与条件变量

7.2.1 互斥锁

互斥锁保证同一时刻只有一个线程访问临界区。

7.2.2 条件变量

条件变量用于线程间的等待与通知。

7.3 异步编程

7.3.1 async 与 await

```
async fn fetch() -> Data {
    let r = request().await;
    r.body
}
```

7.3.2 Future 与 Promise

概念	说明
Future	表示未来的值
Promise	用于完成 Future

第八章 网络编程

8.1 套接字

8.1.1 TCP 套接字

```
let s = TcpStream::connect("127.0.0.1:8080");
s.write(b"hello");
```

8.1.2 UDP 套接字

UDP 套接字无连接，适合低延迟场景。

8.2 HTTP 客户端

8.2.1 发送请求

```
let resp = http::get("https://example.com");
```

8.2.2 处理响应

响应体可按流式读取，避免一次性占用内存。

8.3 服务器开发

8.3.1 简单服务器

```
let l = TcpListener::bind("0.0.0.0:8080");
for c in l.incoming() { handle(c); }
```

8.3.2 路由与中间件

概念	说明
路由	按路径分发请求
中间件	横切逻辑，如日志、鉴权

第九章 测试与调试

9.1 单元测试

9.1.1 测试框架

```
#[test]
fn it_works() {
    assert_eq!(2 + 2, 4);
}
```

9.1.2 断言

常用断言：`assert_eq!`、`assert_ne!`、`assert!`。

9.2 调试技巧

9.2.1 调试器使用

调试器支持断点、单步与变量查看。

9.2.2 日志调试

> 生产环境应关闭 debug 级别日志，避免性能损失。

9.3 性能分析

9.3.1 性能剖析

工具	用途
`perf`	CPU 采样
`flamegraph`	火焰图

9.3.2 内存分析

内存分析可定位泄漏与过度分配。

第十章 部署与运维

10.1 构建发布版本

10.1.1 构建配置

```
[profile.release]
opt-level = 3
lto = true
```

10.1.2 交叉编译

交叉编译需指定目标三元组。

10.2 容器化部署

10.2.1 Docker 镜像

```
FROM scratch
COPY app /app
ENTRYPOINT ["/app"]
```

10.2.2 编排与调度

概念	说明
Pod	最小调度单元
Service	稳定访问入口

10.3 监控与告警

10.3.1 指标采集

指标采集周期建议为 15 秒。

10.3.2 告警规则

> 告警应避免噪声，仅对可操作事件触发。

附录

A 错误码速查表

错误码	含义
E0001	语法错误
E0002	类型不匹配
E0003	未定义标识符

B 命令行参考

命令	说明
`nasm build`	构建项目
`nasm run`	运行项目
`nasm test`	运行测试

C 配置项索引

配置项	默认值
`log.level`	`info`
`log.file`	标准输出

D 常见问题 FAQ

> 问：安装后找不到 `nasm` 命令？
>
> 答：请将安装目录的 `bin/` 加入 `PATH` 环境变量。