

考鼎码(EC2)可实现功能清单

本清单参考 [在线文档](#) (下文简称 参考文档), 时间版本 2024/13/3-18:00, 分析其提及的各个需求, 得出的可实现功能的详细效果清单。

总述

原文:

可执行考鼎码 (Executable Coding Code, EC2) 是一种主要用于少儿信息学启蒙的编程语言; 在不引起混淆的情况下可简称考鼎码。

考鼎码的主要特征为:

解释型脚本编程语言。

支持汉字关键词及命名。

弱类型。

基础类型: 未定义、空、布尔、整数 (系统位宽一致的整数)、浮点数、字节、字符、字节序、字符串。

容器类型: (通用) 序列、映射。

跨平台; 优先支持 WASM 平台。

中期规划:

增加对基础类型任意整数 (任意精度整数) 的支持。

远期规划

增加对任意浮点 (任意精度浮点数) 的支持。

增加对容器类型集合的支持。

提供迭代器、生成器、匿名函数等的支持。

开发基础共识:

1. 使用 wasm 技术, 提供可在浏览器环境下运行的PlayGround程序, 并提供 基本的**代码编辑**功能
2. EC2的实现利用洛书 (Losu) 作为基准语言进行改造, 改造程度应在**合理范围**以内

代码编辑:

1. UI元素包括: 代码编辑框、结果显示区域、运行按钮、清空按钮
2. 代码编辑框:
 1. 关键词高亮
 2. 行号显示
 3. 快捷输入 (基于提示框与鼠标操作逻辑为主)
 4. 语法错误提示
3. IO操作: 基于浏览器图形界面的IO操作, 如输入框

合理范围:

1. 保持基准语言的基本架构与执行机制
2. 保留基准语言的基本数据结构与逻辑系统, 允许非破坏性改造
3. 保留基准语言的语义结构、代码生成体系与虚拟机结构;

本清单范围:

1. EC2的主要特征
2. EC2中期规划特性

- 3. EC2远期特性
- 4. 其他特性

本清单所提及的特性，按可行性分类如下：

- 1. **完全可行**：可以完全按照**参考文档**进行实现
- 2. **可行**：可以根据**参考文档**近似实现，但需要一定的微调
- 3. **可选**：无法按照**参考文档**提供的形式实现，但拥有功能相同或相近的替代实现
- 4. **实验性**：不具备同质性功能，需要添加实验性代码，不能保证最终效果
- 5. **不可行**：过于复杂、违反基本共识或超出能力范围等原因而无法实现的特性；

考鼎码(EC2)可实现功能清单

总述

具体内容

- 1. 程序结构
 - 原文：
 - 方案：
 - 1.1 （可行）
- 2. 考鼎码命名及书写规范
 - 原文：
 - 方案：
 - 2.1 （完全可行）
- 3. 考鼎码关键词
 - 原文
 - 方案
 - 3.1 （可行）
- 4. 考鼎码函数
 - 原文
 - 方案
 - 4.1 （完全可行）
- 5. 运算符
 - 原文
 - 方案
 - 5.1 算数运算符（完全可行）
 - 5.2 赋值运算符（完全可行）
 - 5.3 逻辑运算符（完全可行）
 - 5.4 移位运算符（可选）
- 6. 数字
 - 原文
 - 方案
 - 6.1 系统浮点数（完全可行）
 - 6.2 系统整数（可行）
 - 6.3 任意整数与任意浮点数（实验性）
- 7.字符串
 - 原文
 - 方案
 - 7.1 字节（实验性）
 - 7.2 字符（实验性）
 - 7.3 字节序列与字符串（可选）
- 8.序列
 - 原文
 - 方案

8.1 序列的设计（可行）
9. 映射
原文
方案
9.1映射的设计（可行）
10. 解释器要求
原文
方案
10.1 运行时行为（可行）
10.2 系统功能（实验性）
10.3 中英文关键词（可行）
10.4 内置函数（可行）

具体内容

1. 程序结构

原文：

- `算始/算终` 定义的代码为单个考鼎码程序的唯一入口。
- 算法的参数应由解释器提示用户输入并传入，类型为字符串，可执行反序列化转换为复杂对象，如序列、映射等。
- 算法的返回值应视作程序的返回值，按可执行反序列化的格式输出。

方案：

1.1（可行）

- `算始/算终` 定义的代码为单个考鼎码程序的唯一入口。
- 算法的参数应由用户在提示框输入，类型为字符串，可执行反序列化转换为复杂对象，如序列、映射等。
- 算法的返回值应视作程序的返回值，按可执行反序列化的格式输出。

2. 考鼎码命名及书写规范

原文：

- 函数名、算法名、变量名只能以拉丁字母、下划线、汉字开头，可包含拉丁字母、数字、下划线和汉字。
- 命名不可和保留的关键词、内置函数冲突。
- 大小写敏感；括号、逗号、运算符等使用半角字符。
- 词元分隔符：半角空格、制表符、运算符以及括号（`()`、`[]`、`{}`）、逗号（`,`）等。
- 汉字指 UniHan 定义的汉字，不含标点符号以及全角字母等。
- 根据嵌套关系要求缩进书写，至少两个空格。
- 使用行尾的 `#` 定义行注释。

方案：

2.1（完全可行）

- 函数名、算法名、变量名只能以拉丁字母、下划线、汉字开头，可包含拉丁字母、数字、下划线和汉字。
- 命名不可和保留的关键词、内置函数冲突。
- 大小写敏感； 括号、逗号、运算符等使用半角字符。
- 词元分隔符：半角空格、制表符、运算符以及括号（`()`、`[]`、`{}`）、逗号（`,`）等。
- 汉字指 UniHan 定义的汉字，不含标点符号以及全角字母等。
- 根据嵌套关系要求缩进书写，至少两个空格。
- 使用行尾的 `#` 定义行注释。
- 统一使用 utf-8编码

3. 考鼎码关键词

原文

关键词	解释	示例用法
算始/算终	定义一个算法的开始/终止	
函始/函终	定义一个函数的开始/终止	
若始/若终	定义一个判断的开始/结束	若始 <code>(x > 0)</code>
又若	定义一个判断的其他条件	又若 <code>(x == 0)</code>
若否	定义一个判断的否则条件	
当始/当终	定义一个循环的开始/结束	当始 <code>(x > 0)</code>
返回	返回一项数据	返回 <code>0</code>
真/假	逻辑真/假值	返回 <code>真</code>
非	逻辑非	若始 <code>(非 x > 1)</code>
且/或	逻辑与/或	若始 <code>(x > 1 且 y > 0)</code>
未定义	未定义	返回 <code>未定义</code>
空	空数据	返回 <code>空</code>
导入	导入指定函数库	导入 <code>数学库</code>
声明	声明外部函数或全局变量	声明 <code>sqrt(x)</code>

方案

3.1（可行）

关键词	解释	示例用法
算始/算终	定义一个算法的开始/终止	
函始/函终	定义一个函数的开始/终止	
若始/若终	定义一个判断的开始/结束	若始 (x > 0)
又若	定义一个判断的其他条件	又若 (x == 0)
若否	定义一个判断的否则条件	
当始/当终	定义一个循环的开始/结束	当始 (x > 0)
返回	返回一项数据	返回 0
真/假	逻辑真/假值	返回 真
非	逻辑非	若始 (非 x > 1)
且/或	逻辑与/或，支持短路运算	若始 (x > 1 且 y > 0)
未定义	未定义，基于逻辑伪实现	返回 未定义
空	空数据	返回 空
导入	导入指定函数库	导入 数学库
声明	声明外部函数	声明 sqrt(x)
令	声明变量	令 甲 = 1; 令 乙

- 1. 洛书采用非空（洛书类型：null）为真，空为伪的逻辑模式，可进行以下调整：
 - 1. 新增真类型，使用 未定义 作为逻辑伪
 - 2. 新增空类型，区别于未定义类型
- 2. 导入与声明关键词不包括实际内容
- 3. 修改文法规则，取消段落缩进

4. 考鼎码函数

原文

- 考鼎码使用 函始/函终 两个关键词定义一个函数。
- 每个函数应该具有唯一的名称。
- 函数名称之后用括号指定一个或多个参数名（形参），多个参数名用逗号隔开；若函数不需要参数，则不需要括号和形参列表。
- 调用函数时，函数名然后是小括号包围的实参列表；若函数不需要参数，则括号为空。

- 根据函数的原型，未传递对应的实参时，一律视作传入 `未定义` 处理。

```
# 该函数计算并返回两个数的绝对值之和
函始 绝对值之和 (a, b)
    若始 (a < 0)
        a := -a
    若终

    若始 (b < 0)
        b := -b
    若终

    返回 a + b
函终
```

方案

4.1（完全可行）

- 使用 `函始`/`函终` 两个关键词定义一个函数。
- 每个函数应该具有唯一的名称。
- 函数名称之后用括号指定一个或多个参数名（形参），多个参数名用逗号隔开；若函数不需要参数，则不需要括号和形参列表。
- 调用函数时，函数名然后是小括号包围的实参列表；若函数不需要参数，则括号为空。
- 根据函数的原型，未传递对应的实参时，一律视作传入 `未定义` 处理。
- 函数支持返回多个返回值，不同返回值使用 `,` 隔开

```
函数 交换(a,b)
    返回 b,a
函终
令 甲,乙 = 交换(1,2)
```

5. 运算符

原文

算数运算符、赋值运算符、逻辑运算符、移位运算符

名称	运算符	示例	结果
加法	<code>+</code>	<code>3 + 5</code>	<code>8</code>
减法	<code>-</code>	<code>5 - 3</code>	<code>2</code>
乘法	<code>*</code>	<code>5 * 3</code>	<code>15</code>
除法	<code>/</code>	<code>5 / 2</code>	<code>2.5</code>
整除	<code>//</code>	<code>5 // 2</code>	<code>2</code>

名称	运算符	示例	结果
取模	<code>%</code>	<code>5 % 2</code>	<code>1</code>

名称	运算符	示例	结果
赋值	<code>:=</code>	<code>x := 3</code>	<code>x</code> 的值为 <code>3</code>
赋值	<code>=</code>	<code>x = 3</code>	<code>x</code> 的值为 <code>3</code>

名称	运算符	示例	结果
等于	<code>==</code>	<code>3 == 5</code>	假
大于	<code>></code>	<code>5 > 3</code>	真
小于	<code><</code>	<code>5 < 3</code>	假
大于等于	<code>>=</code>	<code>5 >= 3</code>	真
小于等于	<code><=</code>	<code>5 <= 3</code>	假
不等于	<code>!=</code>	<code>5 != 3</code>	真

名称	运算符	示例	结果
位与	<code>&</code>		
位或	<code> </code>		
位非	<code>~</code>		
位亦或	<code>^</code>		
左移	<code><<</code>		
右移	<code>>></code>		

方案

5.1 算数运算符（完全可行）

5.2 赋值运算符（完全可行）

5.3 逻辑运算符（完全可行）

5.4 移位运算符（可选）

无法直接添加移位运算符，可选择使用函数实现，函数名与运算符名称保持一致

6. 数字

原文

EC2 数字类包括系统浮点数，系统整数、任意整数（任意精度）、任意浮点数四类，其中后两位为中期及以后规划

方案

6.1 系统浮点数（完全可行）

- 1. 系统浮点数是洛书number类型实现
- 2. 支持使用字面量创建

6.2 系统整数（可行）

- 1. 系统整型基于 `int64_t` 进行实现，分配新数据类型 `int`
- 2. 使用 `整数()` 函数进行创建与转化
- 3. 支持 基本的算数运算符，算数运算默认转为浮点类型

6.3 任意整数与任意浮点数（实验性）

- 1. 保留数据类型接口、相应操作函数、内置标识符
- 2. 不进行具体的功能实现

7. 字符串

原文

- 字节：使用 `uint8_t`，使用单引号定义
- 字符：使用 `uint32_t`，使用 `\` 定义
- 字节序列：
 - 使用 `bb` 前缀用二进制定义字节序字面量（可在中间添加 `.` 作为分隔符）。
 - 使用 `bx` 前缀用十六进制定义字节列字面量。
 - 使用 `b64` 前缀用Base64编码定义字节序列字面量
 - 支持 `+`、`[]`、`分割()`
- 字符串
 - 使用 UTF-8 编码的字符序列；使用连续内存存储，可线性访；但使用 `\0` 字符作为终止字符，且只能包含合法的 Unicode 字符。
 - 使用双引号 (`"`) 定义字符串字面量
 - 可通过索引值访问字符串中的每个字节。

操作	运算符/函数	示例	结果
定义字符串	<code>:=</code>	<code>str := "一个字符串"</code>	<code>str</code> 是一个 UTF-8 编码的字节序列。

操作	运算符/函数	示例	结果
获取长度	<code>长度()</code>	<code>len := 长度(str)</code>	<code>len</code> 的值为 <code>15</code> (字节数)。
串接两个字符串	<code>+</code>	<code>str2 := "一" + "个"</code>	<code>str2</code> 的值为 <code>"一个"</code> 。
转换为字符	<code>字符()</code>	<code>uc := 字符(str)</code>	<code>uc</code> 的值为 <code>0x4E00</code> 。
转换为字节	<code>字节()</code>	<code>byte := 字节(str)</code>	<code>byte</code> 的值为 <code>0xE4</code> 。
获取字符数量	<code>.字符数()</code>	<code>nr_ucs := str.字符数()</code>	<code>nr_ucs</code> 的值为 <code>5</code> 。
转换为字符序列	<code>.字符序列()</code>	<code>ucs := str.字符序列()</code>	<code>ucs</code> 为 <code>[\一, \个, \字, \符, \串]</code> 。
分割	<code>.分割()</code>	<code>strings := str.分割()</code>	<code>strings</code> 为 <code>["一", "个", "字", "符", "串"]</code> 。

方案

7.1 字节（实验性）

1. 基于 整数 (int64_t) 实现, 限制为 uint8_t, 支持常规算数运算
2. 使用 `字节()` 函数定义与转换


7.2 字符（实验性）

1. 基于整数实现, 限制为 unit32_t
2. 使用 `字符()` 函数定义与转换

7.3 字节序列与字符串（可选）

1. 使用单引号或双引号字面量定义
2. 一体多用设计: 字面量中可以存在 `\0` 字符, 其在字符串模式下会截断, 但在虚拟机中将其视作字节序列进行存储
3. 支持线性访问, 访问返回值是其对应的ASCII值
4. 支持转义字符
 1. 采用 `\[字母]` 格式类C风格转义字符, 包括
 - `\a`
 - `\b`
 - `\f`
 - `\n`
 - `\r`

- `\t`
- `\v`
- `\\`
- `\"`
- `\'`

- 2. 采用 `\`[十进制 ASCII 码] 格式的转义字符，例如 `\10`
 - 3. 采用 `%`[8bit 16进制 ASCII 码] 格式的转义字符，例如 `%0A %0a`
 - 4. 采用 `\u`[16进制 utf-8 码] 格式的转义字符，例如 `\uf09f8e89` ()
5. 支持基本操作方法

操作	运算符/函数	示例	结果
定义字符串	<code>:=</code>	<code>str := "一个字符串"</code>	<code>str</code> 是一个 UTF-8 编码的字节序列。
获取长度	<code>长度()</code>	<code>len := 长度(str)</code>	<code>len</code> 的值为 <code>15</code> （字节数）。
串接两个字符串	<code>&</code>	<code>str2 := "一" & "个"</code>	<code>str2</code> 的值为 <code>"一个"</code> 。
转换为字符	<code>字符()</code>	<code>uc := 字符(str)</code>	<code>uc</code> 的值为 <code>0x4E00</code> 。
转换为字节	<code>字节()</code>	<code>byte := 字节(str)</code>	<code>byte</code> 的值为 <code>0xE4</code> 。
获取字符数量	<code>字符数()</code>	<code>nr_ucs := 字符数(str)</code>	<code>nr_ucs</code> 的值为 <code>5</code> 。
转换为字符序列	<code>字符序列()</code>	<code>ucs := 字符序列(str)</code>	划分为字符数值序列
按字符分割	<code>分割()</code>	<code>strings := 分割(str)</code>	<code>strings</code> 为 <code>["一", "个", "字", "符", "串"]</code> 。

8.序列

原文

- 1. （通用）序列：由一组有序的数据构成，数据项的类型可以是空、布尔、字节、字符、整数、任意整数、浮点、任意浮点、序列、映射等。
- 2. 超范围访问均返回 `未定义`；设置给定索引的元素为 `未定义` 将移除该元素。
- 3. 考鼎码中序列的基本操作：

操作	运算符	示例	结果
定义空序列	<code>:=</code>	<code>s := []</code>	<code>s</code> 是一个空序列

操作	运算符	示例	结果
串接两个序列	<code>+</code>	<code>s := [0, 2] + [3, 4]</code>	<code>s</code> 为 <code>[0, 2, 3, 4]</code>
获取序列中第 <code>i</code> 个元素	<code>[i]</code>	<code>a := s[0]</code>	<code>a</code> 的值为 <code>0</code>
设置序列中第 <code>i</code> 个元素	<code>[i]</code>	<code>s[0] := 1</code>	<code>s</code> 变为 <code>[1, 2, 3, 4]</code>
将某个元素插入到序列的前面	<code>+</code>	<code>s := [0] + s</code>	<code>s</code> 变为 <code>[0, 1, 2, 3, 4]</code>
将某个元素追加到序列的后面	<code>+</code>	<code>s := s + [5]</code>	<code>s</code> 变为 <code>[0, 1, 2, 3, 4, 5]</code>
使用 <code>未定义</code> 移除序列中的元素	<code>:=</code>	<code>s[0] := 未定义</code>	<code>s</code> 变为 <code>[1, 2, 3, 4, 5]</code>

方案

8.1 序列的设计（可行）

- （通用）序列：由一组有序的数据构成，数据项的类型可以是EC2所有被支持的数据类型，序列下标由 0 开始

洛书默认是由 1 开始的，这个可以修改成从 0 开始

- 超范围访问均返回 `未定义`；设置给定索引的元素为 `未定义` 将移除该元素。
- 支持 `[]` 访问元素与赋值
- 支持 `+` 追加操作

9. 映射

原文

- 类似 Python 字典类型，可使用字节、字符、整数、字符串等作为键名。
- 可使用映射方法 `.键序列()` 获取映射中所有键构成的序列。
- 可使用映射方法 `.值序列()` 获取映射中所有值构成的序列

操作	运算符/函数	示例	结果
定义映射	<code>:=</code>	<code>map := {"姓名": "张三", "年龄": 7}</code>	<code>map</code> 是包含两个键值对的映射。
获取长度	<code>长度()</code>	<code>len := 长度(map)</code>	<code>len</code> 的值为 <code>2</code> （键值对个数）。
合并两个映射	<code>+</code>	<code>map := {"姓名": "张三"} + {"年龄": 7}</code>	<code>map</code> 包含两个键值对，如有相同键名，后者覆盖前者。

操作	运算符/函数	示例	结果
获取键序列	<code>.键序列()</code>	<code>keys := map.键序列()</code>	<code>keys</code> 的值为 <code>["姓名", "年龄"]</code> 。
获取值序列	<code>.值序列()</code>	<code>values := map.值序列()</code>	<code>values</code> 为 <code>["张三", 7]</code> 。
设置键值	<code>:=</code>	<code>map["姓名"] := "李四"</code>	<code>map</code> 被改变。
访问键值	<code>[]</code>	<code>name := map["姓名"]</code>	<code>name</code> 的值为 <code>"李四"</code> 。
移除一个键值对	<code>:= 未定义</code>	<code>map["姓名"] := 未定义</code>	<code>map</code> 现在仅包含一个键值对

方案

9.1映射的设计（可行）

1. 类似 Python 字典类型，可使用数字、字符串等作为键名。

```
map := {"姓名": "张三", "年龄": 7}`
```

2. `长度()` 获取长度
3. 可使用方法 `键序列()` 获取映射中所有键构成的序列。
3. 可使用方法 `值序列()` 获取映射中所有值构成的序列。
4. 合并映射 `+`
5. 设置键值与设置键值 `[]`
6. 移除键值对，赋值为 `未定义` 移除键值对（依赖GC功能）

10. 解释器要求

原文

1. 循环应有最大上限控制，超过此上限（默认为 `65535`）的报 `可能的死循环` 错误。
2. 函数的调用栈帧应有最大上限（默认为 `65535`）控制，超过此上限的报 `函数调用嵌套过深` 错误。
3. 在 WASM 平台上，可提供上述上限的调整功能。
4. 应统计算法的基础运算次数、函数调用次数等，并在执行结束后输出统计信息。
5. 在 WASM 平台上，`输入()` 函数的实现应通过对话框进行，加上提示信息。
6. 在 WASM 平台上，若算法定义有参数，应通过对话框提示用户输入初始参数。
7. 算法的返回值作为整个程序的返回值按可执行反序列化的形式输出，如 `[真, bxE3D4, "字符串", \字, 'A', '\xA3, 0a12345678900987654321, 12345]``。
8. `执行()` 函数指定的系统功能主要包括：
 9. 拍手：通过 HTML 展示动画（GIF）实现。
 - 说出：通过某个 Text to Speech 引擎实现。

- 10. 可参考 [HVML PurC 解释器](#)提供的变体实现各数据类型。
- 11. 可基于[C++ bigint 类](#)实现对任意精度整数的支持。
- 12. 可考虑同时支持中文及英文关键词

方案

10.1 运行时行为（可行）

- 1. 使用中文错误处理信息，包括语法检查与运行时错误，标注出错位置（行号）
- 2. 使用弱数据类型，算数运算符包括部分强类型属性

部分强类型属性指：针对不同数据类型采取不同的行为，无法隐式转换的类型运算时会触发类型错误

- 3. 循环与函数调用具有上限（上限 65535），可以通过C-API设置；函数最大递归深度受制于虚拟机堆栈大小，一般< 65535
- 4. 统计基础运算次数与函数调用次数：
 - 1. 统计各类运算的执行总次数（基于相应指令执行次数实现）
 - 2. 统计脚本中各函数的调用次数
 - 3. 统计外部函数调用次数（需在外部函数中设置）
- 5. `输入()` 函数的实现应通过对话框进行，加上提示信息。

10.2 系统功能（实验性）

- 1. `执行()` 函数指定的系统功能主要包括：
 - 1. 拍手：通过 HTML 展示动画（GIF）实现。
 - 2. 说出：通过某个 Text to Speech 引擎实现。
- 2. 上述问题主要面临问题为：
 - 1. 洛书尚未有wasm图形库支持，可能无法满足复杂的绘图需求
 - 2. Web Speech API 在不同的浏览器环境下可能存在的异常行为
- 3. 上述需求可选的实现方案：
 - 1. 拍手：使用弹窗完成基本演示
 - 2. 说出：使用Web Speech API 进行语音合成，同时输出 `说出:xxxx`，提示文本

10.3 中英文关键词（可行）

尽可能保留部分原生关键词，其余部分使用中文关键词替代

10.4 内置函数（可行）

内置函数	解释	示例用法
<code>输入()</code>	提示并获取用户输入	<code>x := 输入("姓名")</code>
<code>输出()</code>	输出变量或者数据	<code>输出(x)</code>
<code>整数()</code>	将指定数据转换为系统整数	<code>x := 整数(输入("系统整数"))</code>
<code>字节()</code>	将指定数据转换为字节	<code>x := 字节(输入("任意字母"))</code>

内置函数	解释	示例用法
字符()	将指定数据转换为字符	<code>x := 字符(输入("任意中文"))</code>
浮点()	将指定数据转换为浮点数	<code>x := 浮点(输入("浮点数"))</code>
终止()	终止算法执行	<code>终止()</code>
执行()	执行一个系统动作	<code>执行("拍手")</code>
长度()	返回给定数据的长度或数据项数量	<code>长度(s)</code>
转储()	将给定数据按可执行反序列化的格式格式化	<code>转储([1, 2, 3])</code>
解析()	将给定的字符串执行反序列化操作	<code>解析("[1, 2, 3]")</code>