使用C语言实现Lox脚本语言

Table of Contents

1. 字节码块

> 如果你发现自己几乎把所有的时间都花在了理论上，那就开始把注意力转向实践；它会改进你的理论。如果你发现你几乎把所有的时间都花在了实践上，那就开始把注意力转向理论；它会改善你的实践。
>
> — 高德纳

1.1. 开始

让我们先编写一些基本的 代码 。先从 `main` 函数开始。

main.c, create new file

```
#include "common.h"

int main(int argc, const char* argv[]) { ①
  return 0;
}
```

① `const char*` 表示一个可变指针指向了不可变的字符/字符串。

我们会把常用的一些类型和常量放置在 `common.h` 中。

common.h, create new file

```
#ifndef clox_common_h ①
#define clox_common_h

#include <stdbool.h>
#include <stddef.h>
#include <stdint.h>

#endif
```

① `#ifndef clox_common_h` 表示如果没有定义过 `clox_common_h` ，则定义之。如果定义过，则不执行以上代码片段。

1.2. 指令块

块（chunk）表示字节码序列。

chunk.h, create new file

```
#ifndef clox_chunk_h
#define clox_chunk_h

#include "common.h"

#endif
```

在字节码格式中，每条指令都对应一个单字节的操作码（opcode）。所以才叫字节码。我们先来编写一条最简单的字节码指令 `OP_RETURN` 。这条指令表示“从当前函数返回”。不过现在还不具备这个功能。

chunk.h

```
#include "common.h"

typedef enum {
  OP_RETURN,
} OpCode;

#endif
```

1.2.1. 指令的动态数组

字节码是一系列指令。我们会存储指令和一些其它数据，所以让我们创建一个结构体来保存数据。

chunk.h, add after enum OpCode

```
} OpCode;

typedef struct {
  uint8_t* code; ①
} Chunk;

#endif
```

① `code` 是指向字节数组的开头位置的指针。

由于我们不知道字节数组的具体大小，所以需要使用动态数组。动态数组有以下特点：

- 对缓存友好，因为是紧挨着存储的。
- 通过数组索引查找元素是常数时间复杂度。
- 在数组末尾追加元素是常数时间复杂度。

动态数组其实就是Java中的 `ArrayList` 数据类型。在C语言中需要我们自己来实现。

chunk.h, in struct Chunk

```
typedef struct {
  int count;    ①
  int capacity; ②
  uint8_t* code;
} Chunk;
```

① 数组中已经使用的数量

② 数组的容量（大小）

创建一个实例化 Chunk 的接口：

chunk.h, add after struct Chunk

```
} Chunk;

void initChunk(Chunk* chunk);

#endif
```

然后实现接口：

chunk.c, create new file

```
#include <stdlib.h>

#include "chunk.h"

void initChunk(Chunk* chunk) {
  chunk->count = 0;
  chunk->capacity = 0;
  chunk->code = NULL;
}
```

动态数组的初始状态是空数组。我们还没有分配一个数组出来。为了可以将一个字节追加到块的末尾，我们需要一个新的接口。

chunk.h, add after initChunk()

```
void initChunk(Chunk* chunk);
void writeChunk(Chunk* chunk, uint8_t byte);

#endif
```