V8 - 基础（上）

目录

基本概念

Pre - 基本概念

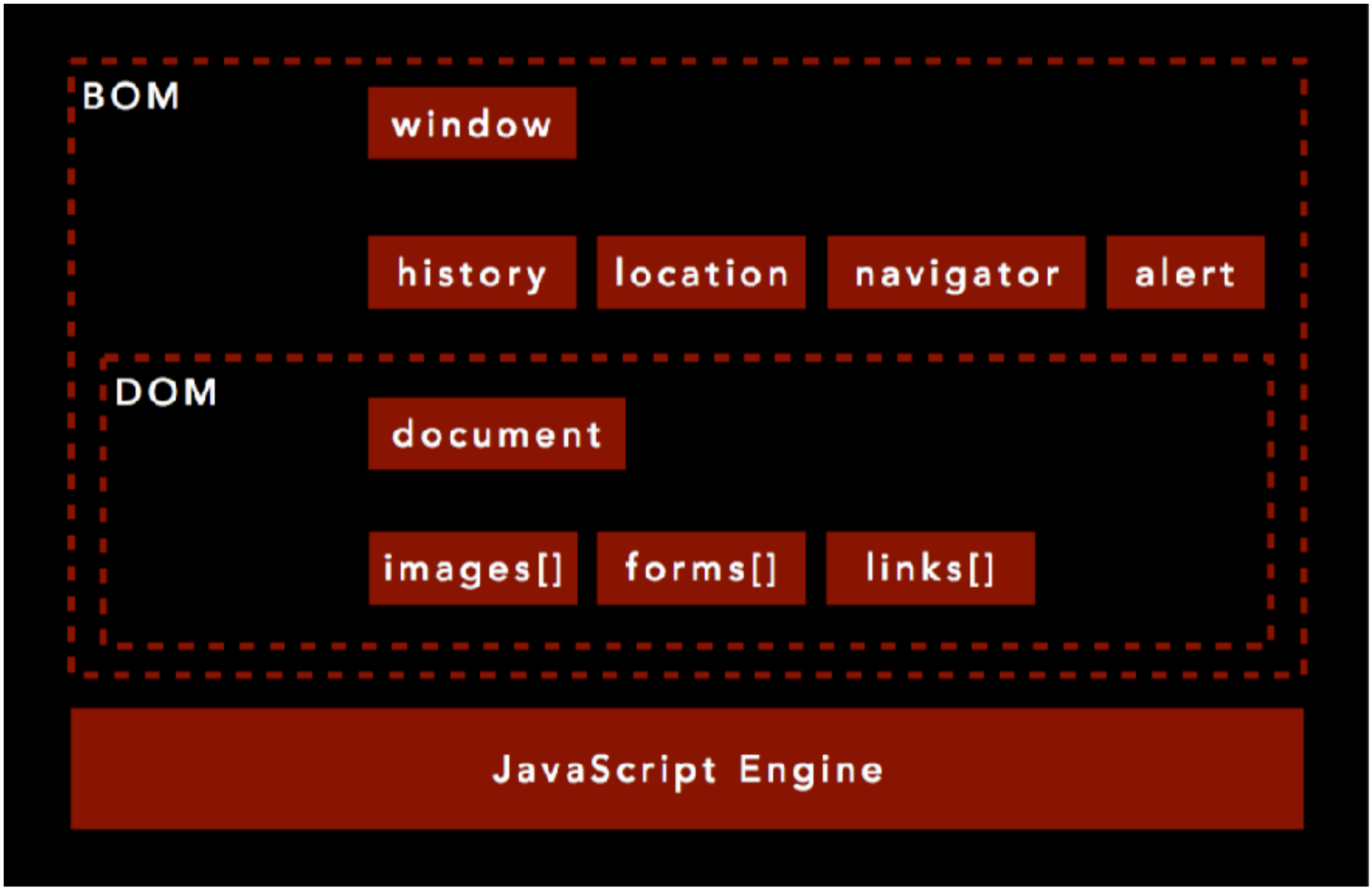
```
[
    'window',
    'document',

    'eval',
    'isNaN',
    'parseInt',
    'parseFloat',
    'decodeURIComponent',
    'encodeURIComponent',
    'setTimeout',
    'setInterval',

    'Object',
    'Function',
    'Boolean',
    'Error',
    'Number',
    'Math',
    'Date',
    'String',
    'RegExp',
    'Array',

    'XMLHttpRequest',
    'JSON',
    'Promise',
    'WebAssembly',
]
```

基本概念

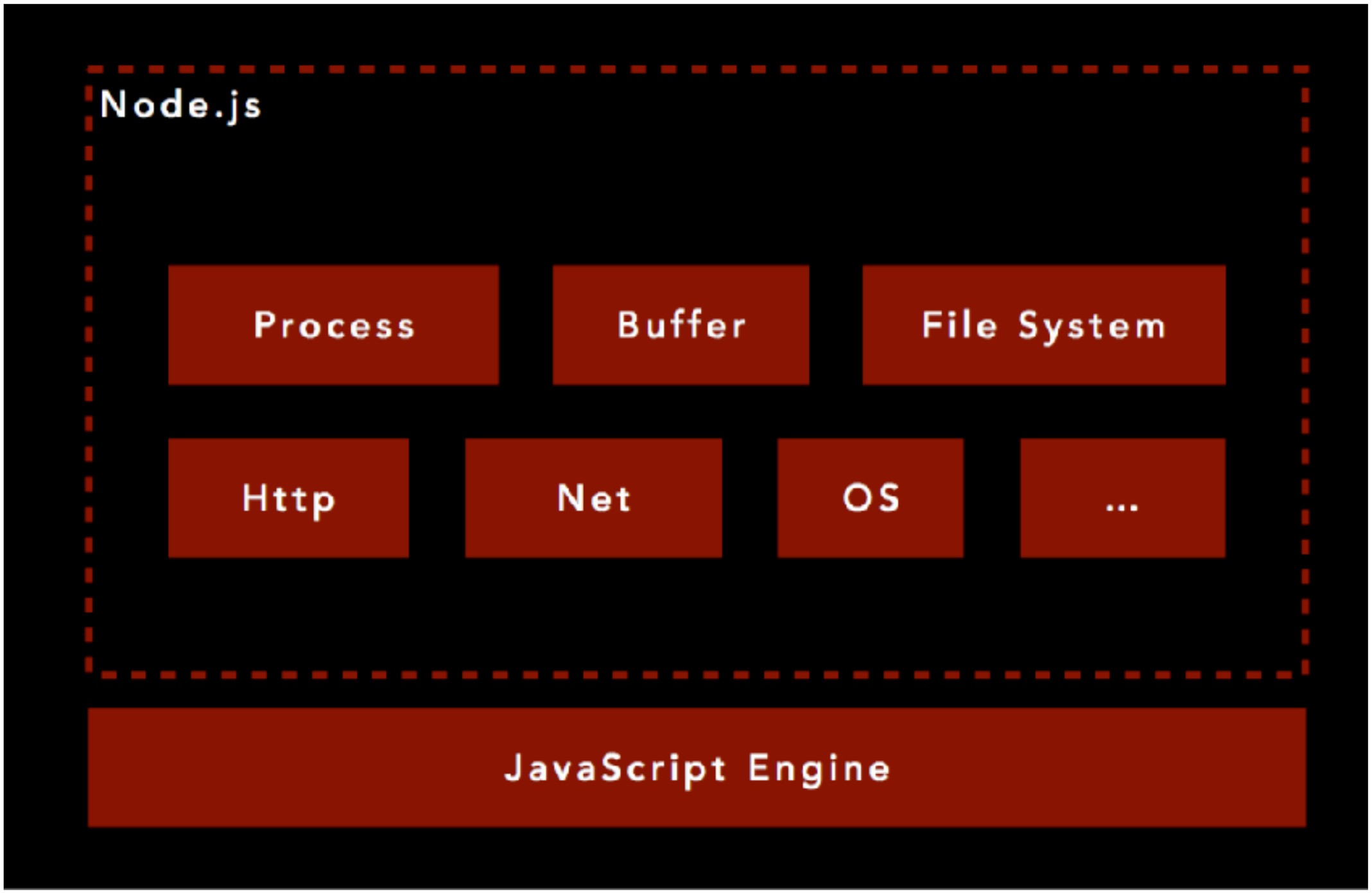
- 浏览器环境
- Node.js 环境

- JavaScript 内置对象

基本概念 - 浏览器环境


BOM
window
history
location
navigator
alert
DOM
document
images[]
forms[]
links[]
JavaScript Engine

基本概念 - Node.js环境



基本概念 - JavaScript内置对象

https://developer.mozilla.org/zh-CN/docs/Web/JavaScript/Reference/Global_Objects

基本概念 - 什么是V8

- JavaScript 引擎 - 一个专门处理JavaScript脚本的虚拟机

内存机制

内存机制

- 新生代内存区
    - From
    - To
- 老生代内存区
    - 老生代指针区
    - 老生代数据区
    - 机器代码区
    - Map区
    - Cell区
    - 大对象区
- GC

内存机制 - 新生代内存区

- 基本的数据对象被分配到这里，区域很小但是垃圾回收的比较频繁
- GC - 复制算法
  - From空间
  - To空间

内存机制 - 老生代内存区

- 老生代指针区：包含很多会指向其他对象的对象，大部分该对象是从新生代内存“晋升”后会移动到这里

- 老生代数据区：包含那些只含有原始数据的对象（不指向到其他的对象的对象）比如: String boxed numbers, arrays of number

- 大数据区：这个空间主要包含的事那些超过其他区域大小限制的对象，每个对象都有自己的内存区域，GC中不会被移动

内存机制 - 老生代内存区

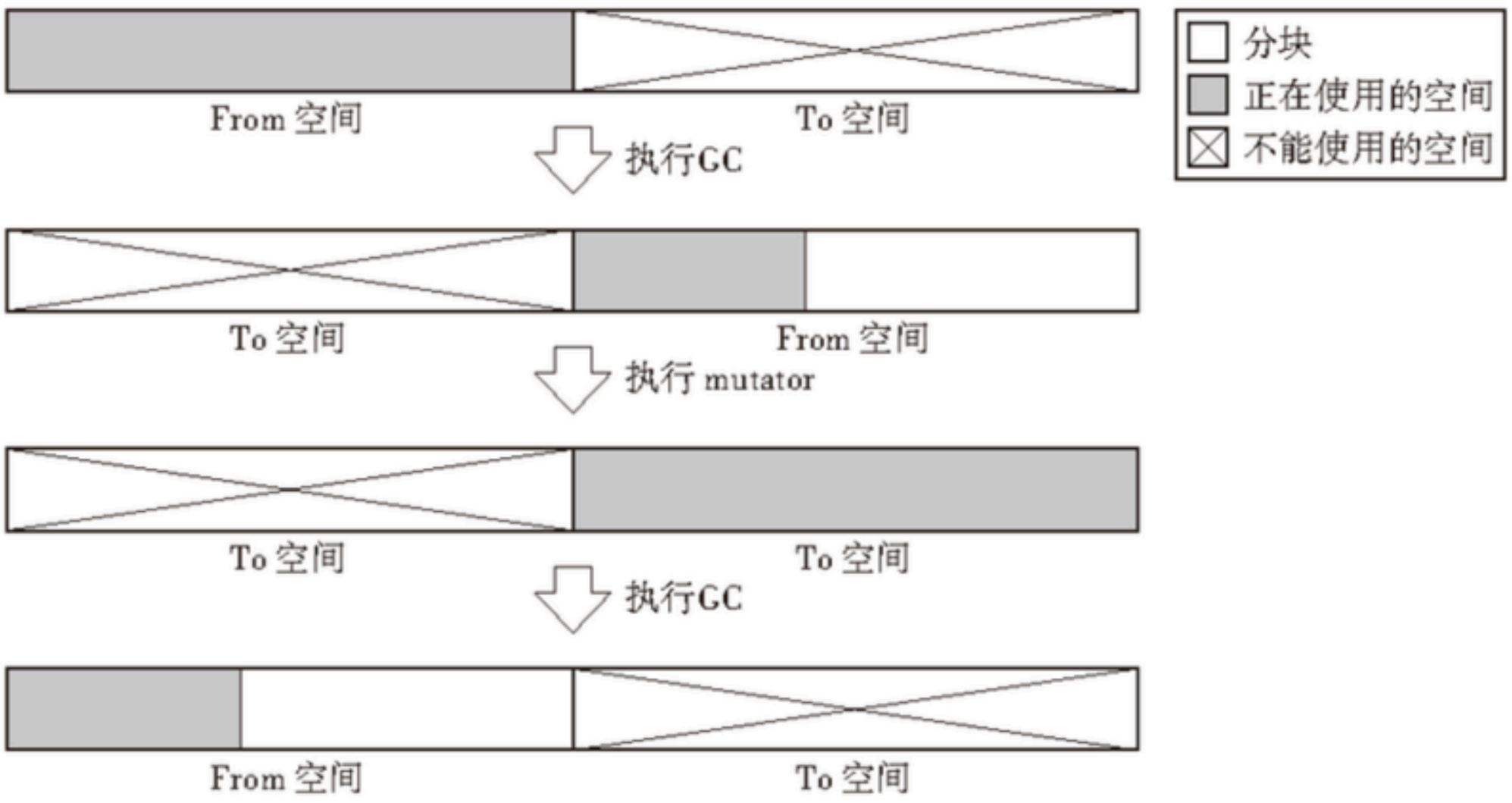
- 代码区：JIT指令的代码对象，唯一可以操作内存的区域
- Cell、属性Cell、Map区：存放一些内存大小相同、结构简单的对象，内置类或者方法等。

内存机制 - GC

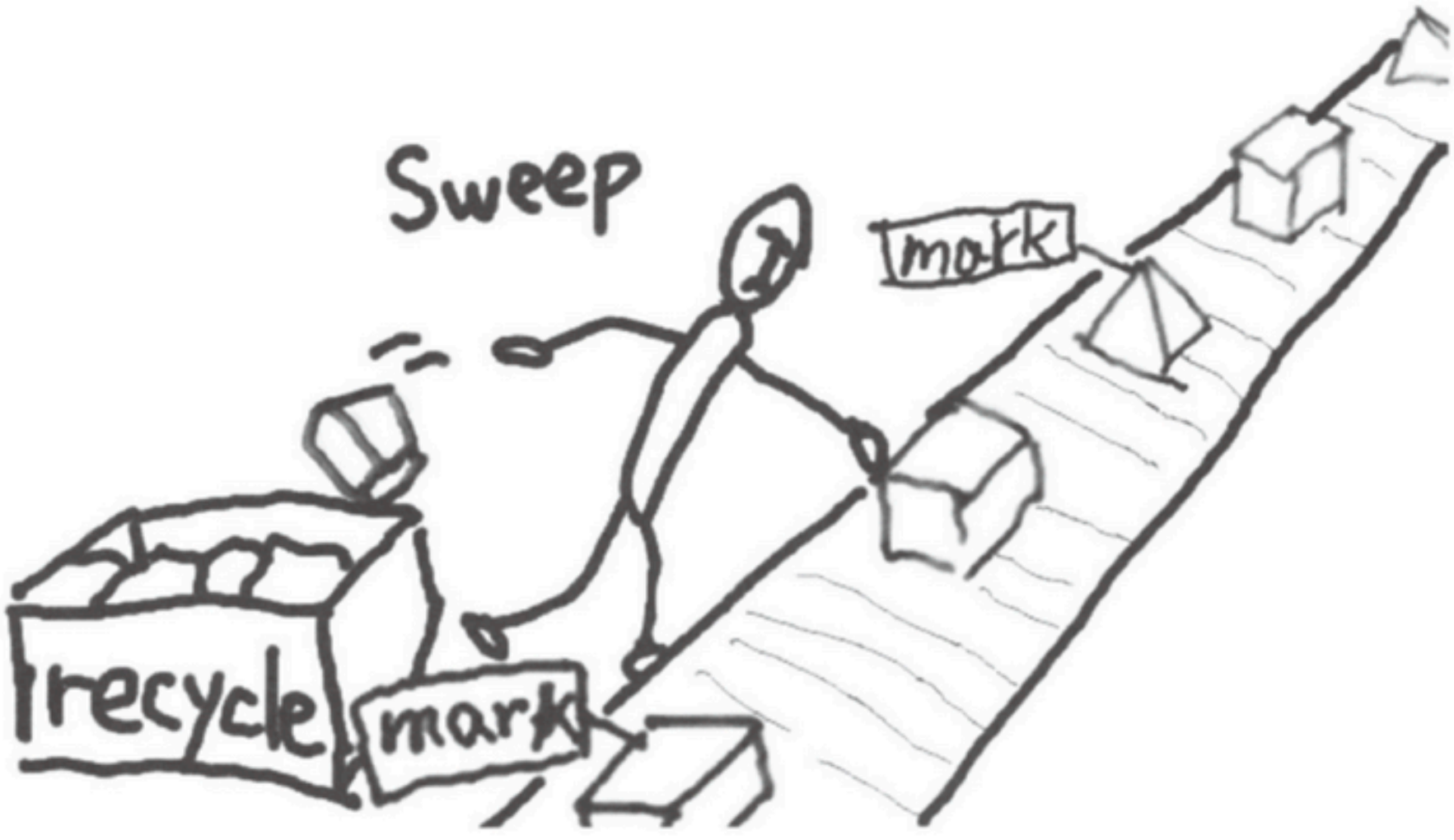
- 新生代区的GC
- 老生代区的GC

内存机制 - GC - 新生代

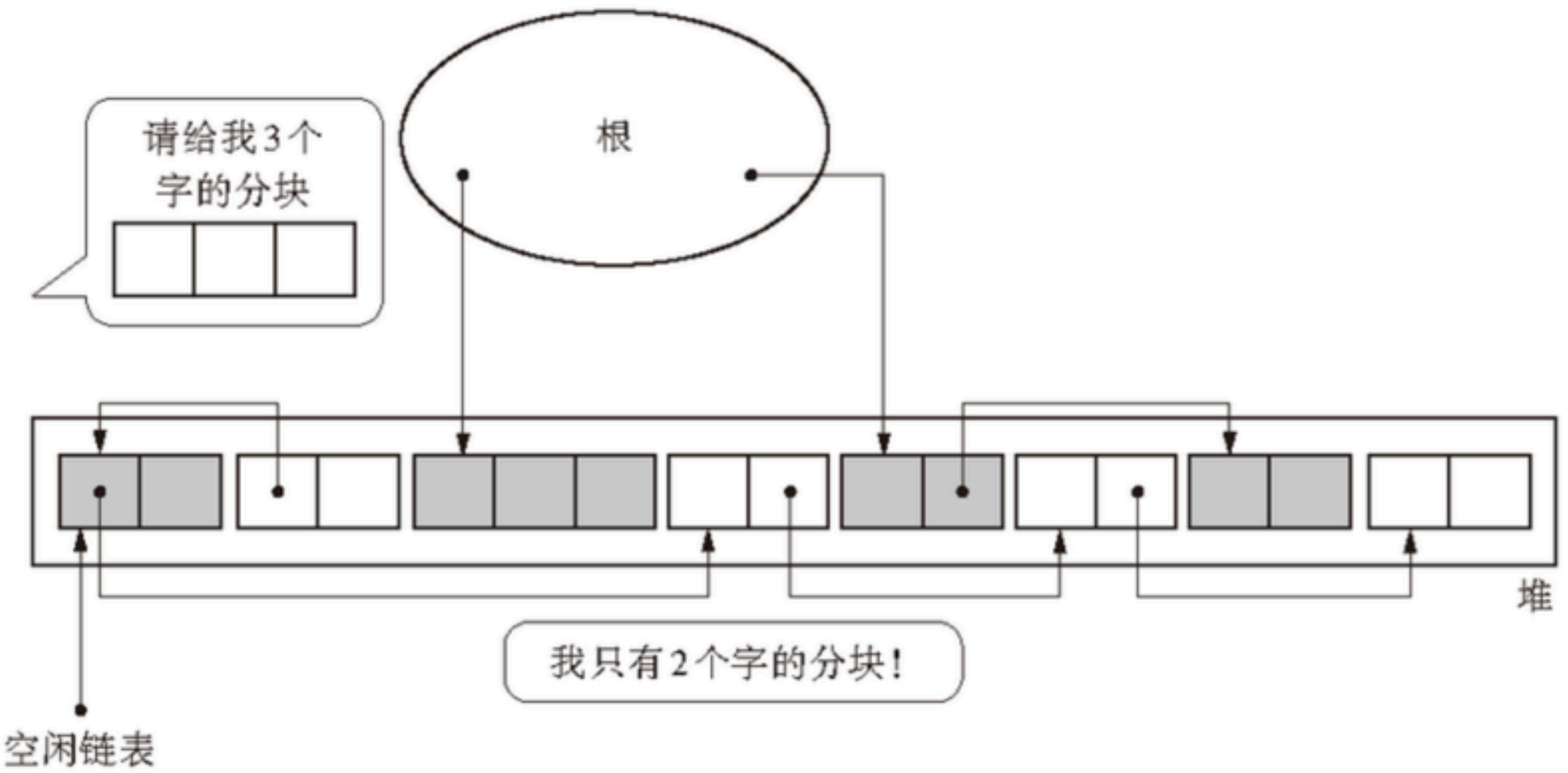
内存机制 - GC - 新生代


From 空间
To 空间
执行GC
To 空间
From 空间
执行 mutator
To 空间
To 空间
执行GC
From 空间
To 空间
分块
正在使用的空间
不能使用的空间

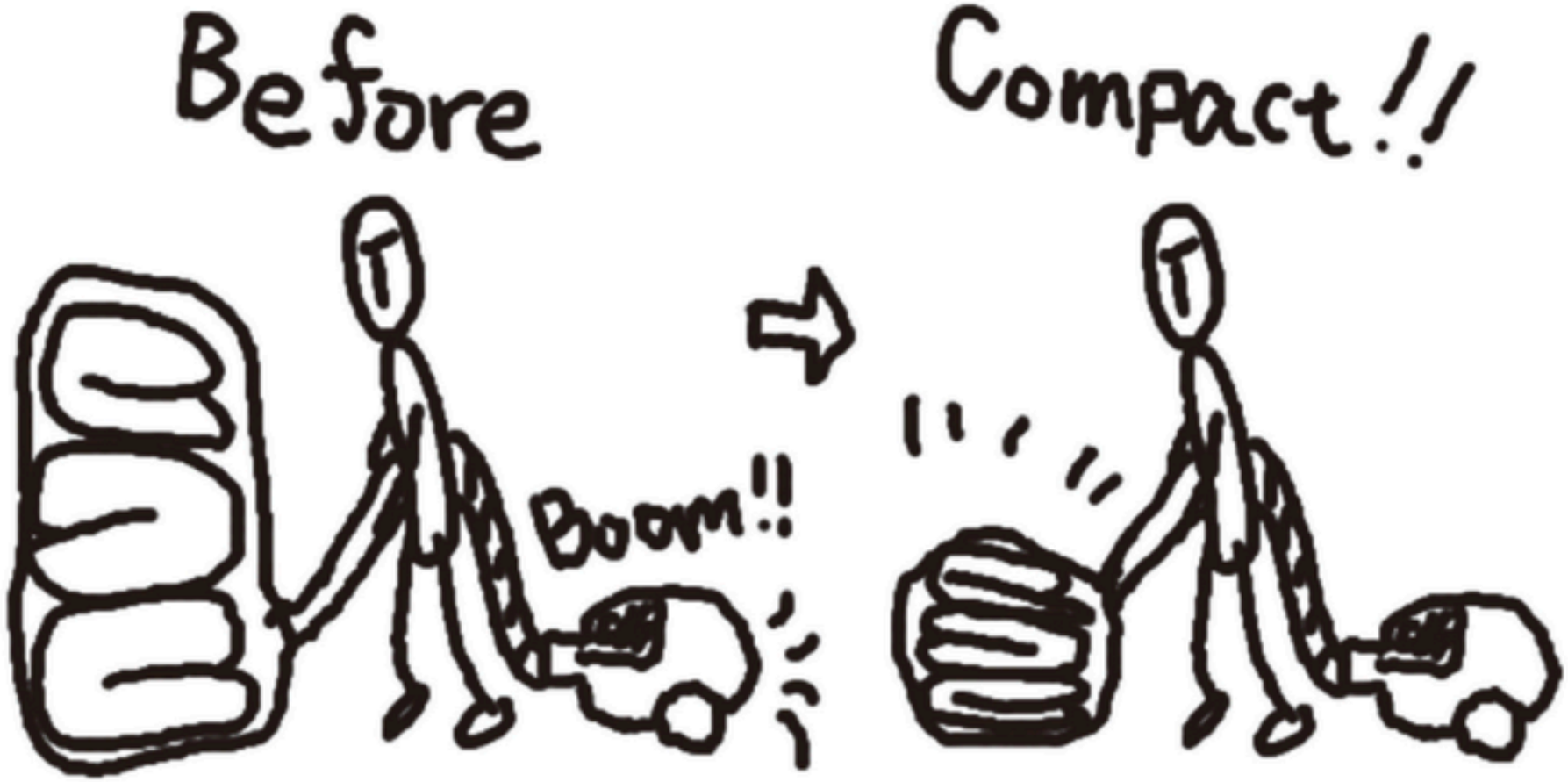
内存机制 - GC - 老生代



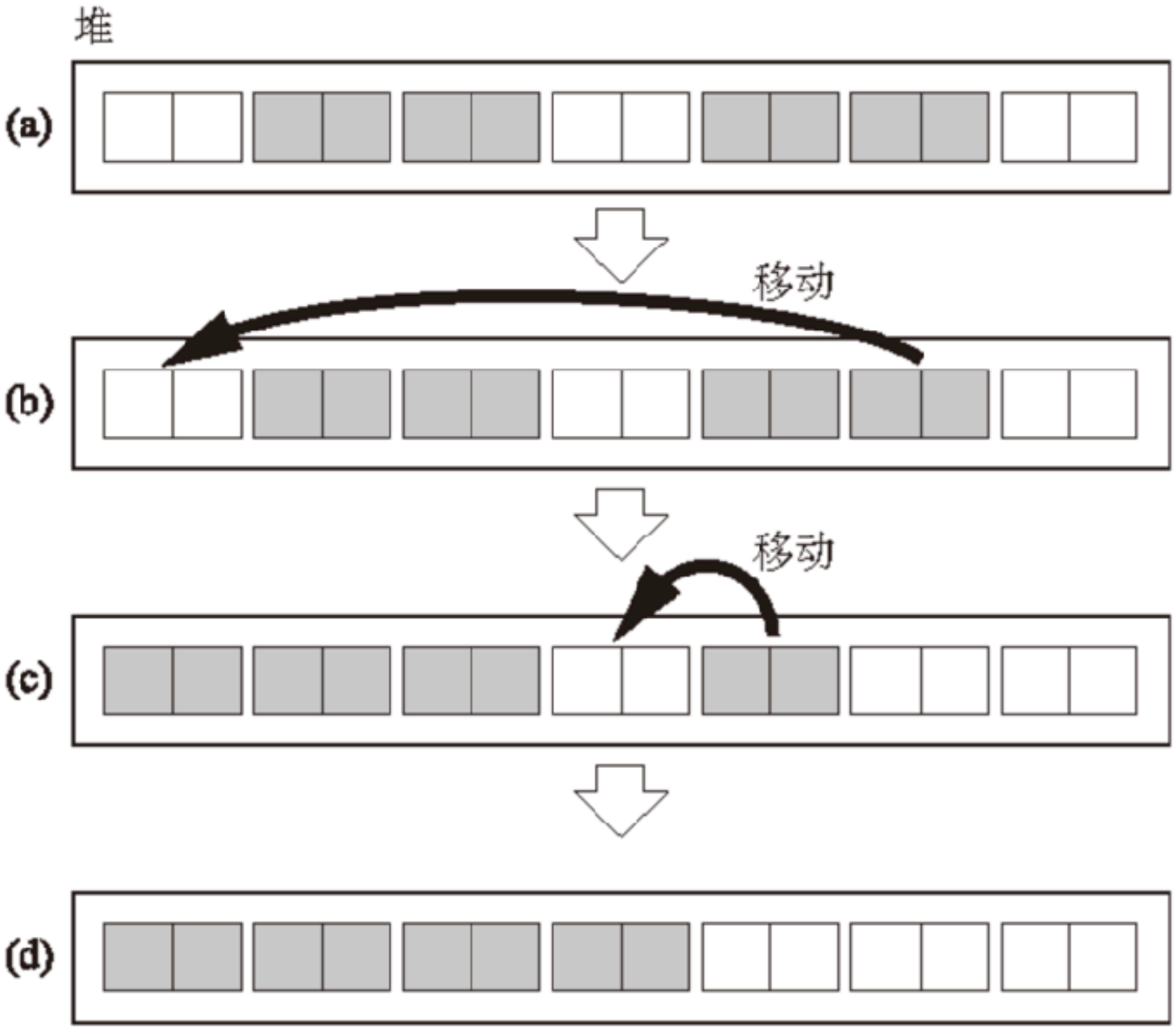
内存机制 - GC - 老生代



内存机制 - GC - 老生代



内存机制 - GC - 老生代



Isolate - 隔离实例

Isolate

- 它代表的是一个V8引擎的实例
- 与别的Isolate实例完全分离，互不干扰
- 不是线程安全的，一个实例不能同时在多个线程中使用
- 在 Node.js 的 C++扩展时，已经处于了 Chrome V8的isolate环境中，直接使用Node.js中的环境即可，不需要再创建

Isolate

```
// Create a new Isolate and make it the current one.
Isolate::CreateParams create_params;
create_params.array_buffer_allocator =
    v8::ArrayBuffer::Allocator::NewDefaultAllocator();
Isolate* isolate = Isolate::New(create_params);
{
```

```
static void RunCallback(const v8::FunctionCallbackInfo<v8::Value>& args) {
  if (args.Length() < 1) return;
  Isolate* isolate = args.GetIsolate();
  HandleScope scope(isolate);

  Local<Function> cb = Local<Function>::Cast(args[0]);
  const unsigned argc = 1;
  Local<Value> argv[argc] = { String::NewFromUtf8(isolate, "hello world") };
  cb->Call(Null(isolate), argc, argv);
}
```

Context - 上下文

Context

- Context用来定义JavaScript执行环境的一个对象，在创建的时候要指明属于哪个实例
- 沙箱化的执行上下文环境，内部预置了一系列的对象和函数 (Node vm模块)

Context

```
// Create a new Isolate and make it the current one.
Isolate::CreateParams create_params;
create_params.array_buffer_allocator =
    v8::ArrayBuffer::Allocator::NewDefaultAllocator();
Isolate* isolate = Isolate::New(create_params);
{
    Isolate::Scope isolate_scope(isolate);

    // Create a stack-allocated handle scope.
    HandleScope handle_scope(isolate);

    // Create a new context.

    Local<Context> context = Context::New(isolate);
```

Script - 腳本

Script

- 包含一段已经编译好的JavaScript脚本的对象， 数据类型就是Script
- 编译时要与一个处于活动状态的上下文进行绑定

Script

```
// Create a new Isolate and make it the current one.
Isolate::CreateParams create_params;
create_params.array_buffer_allocator =
    v8::ArrayBuffer::Allocator::NewDefaultAllocator();
Isolate* isolate = Isolate::New(create_params);
{
    Isolate::Scope isolate_scope(isolate);

    // Create a stack-allocated handle scope.
    HandleScope handle_scope(isolate);

    // Create a new context.

    Local<Context> context = Context::New(isolate);


    // Enter the context for compiling and running the hello world script.
    Context::Scope context_scope(context);

        // Create a string containing the JavaScript source code.
        Local<String> source =
            String::NewFromUtf8(
                isolate, "\n\
\"Hello\" + \", World!\"\n\
");

        // Compile the source code.
        Local<Script> script =
            Script::Compile(context, source).ToLocalChecked();

        // Run the script to get the result.
        Local<Value> result = script->Run(context).ToLocalChecked();


        // Convert the result to an UTF8 string and print it.
        String::Utf8Value utf8(result);
        printf("%s\n", *utf8);

    }
```

Handle - 句柄

Handle

- Handle在V8中是一个非常重要的概念，它提供了对于堆内存中JavaScript数据对象的一个引用
- Handle VS 指针？

Handle

- 为什么不用指针?
    - V8在做垃圾回收的时候，通常会讲JS数据对象移动，如果使用指针的话，容易造成很多野指针，而通过Handle的话，垃圾回收器只需要更新引用了这个数据块的Handle，可以避免该情况

Handle

- **本地句柄 v8::Local**
- **持久句柄 v8::Persistent**
- 永生句柄 v8::Eternal
- 待实本地句柄
- 其他句柄

Handle

- 句柄的本质是C++的一个模版类，根据不同的V8数据类型进行不同的声明 v8::Local<T> v8::Persistent(T)

- v8::Local<v8::Number> 本地JS数值类型句柄

- v8::Persistent<v8::String> 持久JS字符串类型句柄

Handle - Local

- 存在于栈内存中，他们的生命周期是由其所在的HandleScope决定
- HandleScope会在一个函数体内一开始被声明，当一个句柄对象被删除的时候，如果在HandleScope所创建的那些Handle所指的对象没有被其他地方所引用则会被GC掉

Handle - Persistent

- 持久句柄，类似于Local句柄，但是不受HandleScope的管理，其作用域可以延伸到不同的函数当中（DOM节点）

Handle - Eternal

- 在程序的整个生命周期内都是不会被删除的，永生句柄的开销非常小，因为它不需要垃圾回收。

Handle - Maybe Local

```
Local<Value> x = some_value;
Local<String> s = x.ToString();
s->anything();
```

Handle - Maybe Local

```
Local<Value> x = some_value;
Local<String> s = x.ToString();

if (!s.IsEmpty()) {
    s->anything();
}
```

Handle - Maybe Local

```
Local<Value> x = some_value;
Local<String> s = x.ToString(); // 返回Maybe Local

Local<String> _s = s.ToLocalChecked();
```

HandleScope - 句柄作用域

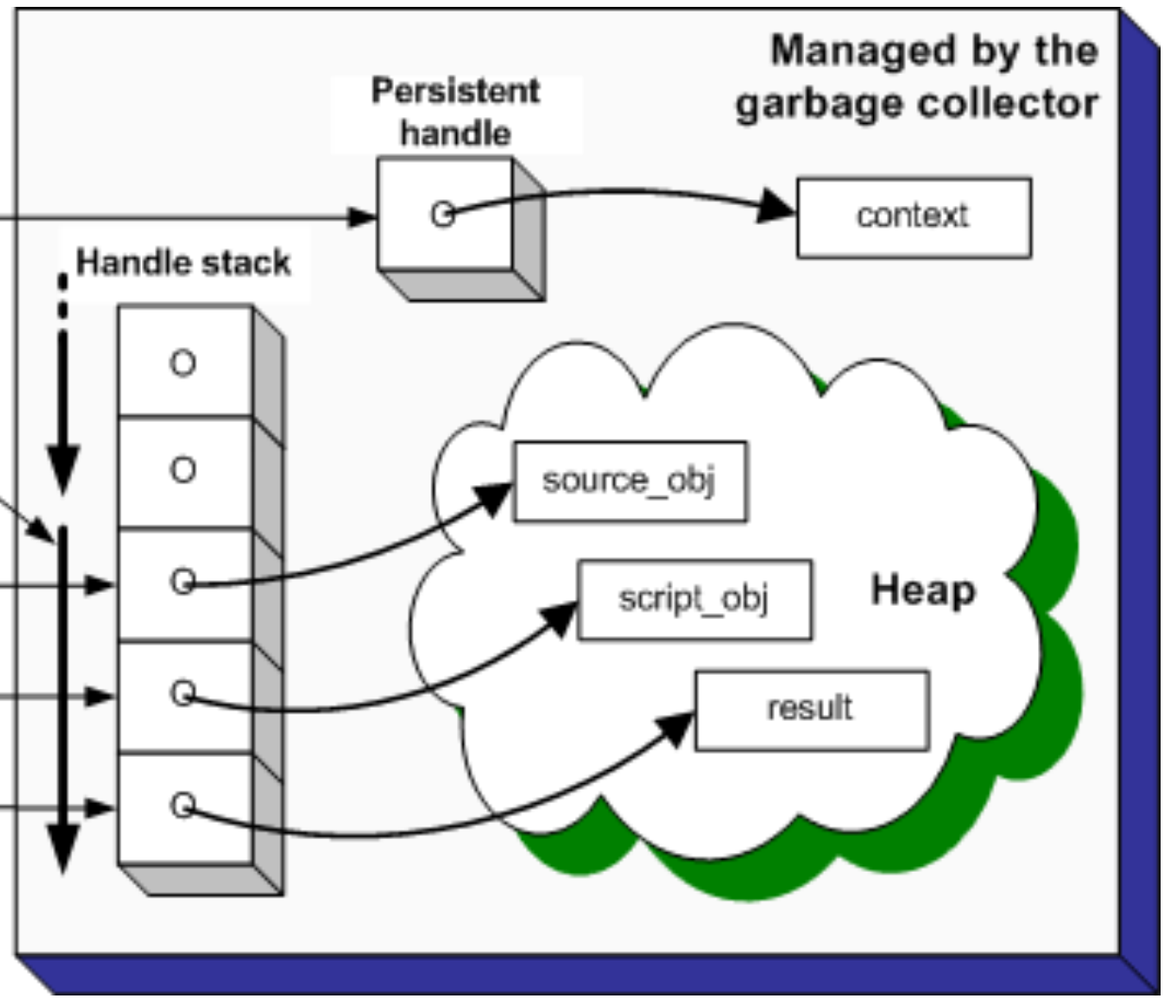
HandleScope

- 本质上是一个维护一堆Handle的容器，统一管理，当一个HandleScope对象的析构函数被调用时，在这个作用域中创建的所有Handle都会从栈中被抹去

HandleScope

hello-world.cc

```
1. Handle<Context> context = Context::New();

   2. Context::Scope context_scope(context);

                3. HandleScope handle_scope;

      4. Handle<String> source_obj = String::New(argv[[]1]);

5. Handle<Script> script_obj = Script::Compile(source_obj);

      6. Handle<Value> local_result = script_obj->Run();

                7. context.Dispose();
```



HandleScope

```
function returnValue() {
  const number = 233;
  return number;
}
```

HandleScope

```
v8::Local<v8::Number> ReturnValue()
{
    v8::Isolate* isolate = v8::Isolate::GetCurrent();
    v8::HandleScope scope(isolate);

    v8::Local<v8::Number> number = v8::Number::New(isolate, 2333);
    return number;
}
```

HandleScope

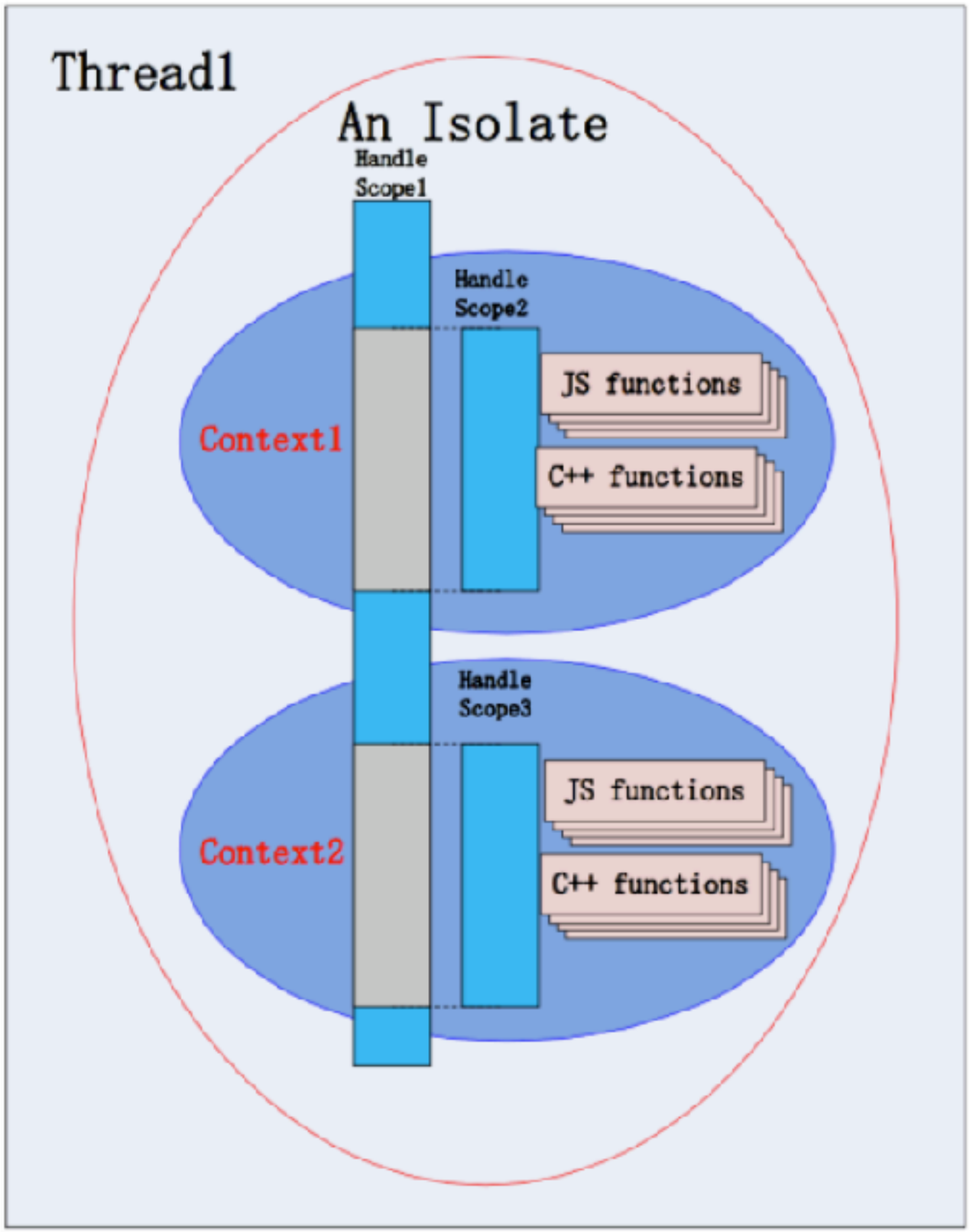
- EscapeHandleScope - 可逃句柄作用域
  - Escape函数，可以给一个Handle“豁免权”，将其复制到一个封闭的作用域内，并且删除其他的Local Handle，然后返回这个新复制的Handle

HandleScope

```
v8::Local<v8::Number> ReturnValue()
{
    v8::Isolate* isolate = v8::Isolate::GetCurrent();
    v8::EscapableHandleScope scope(isolate);

    v8::Local<v8::Number> number = v8::Number::New(isolate, 2333);
    return scope.Escape(number);
}
```

总结

总结


Thread1
An Isolate
Handle
Scope1
Handle
Scope2
JS functions
C++ functions
Context1
Handle
Scope3
JS functions
C++ functions
Context2

QA

Thx