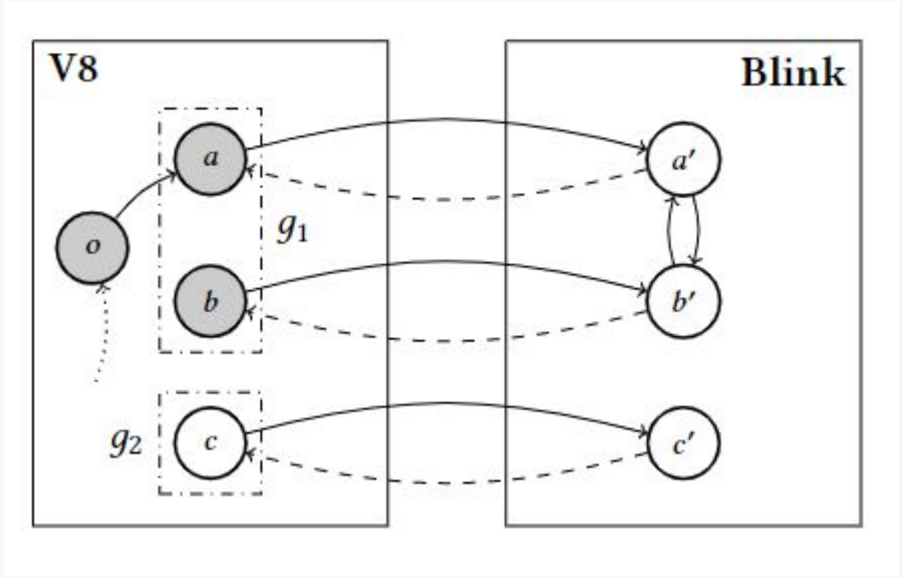
Oilpan: V8中的C++垃圾回收器

PLCT 陆亚涵

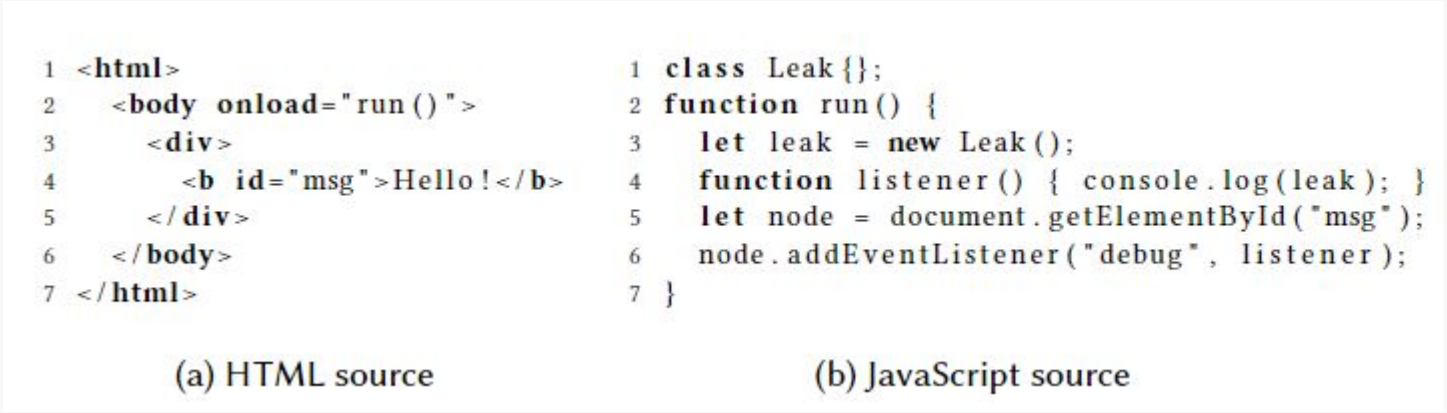
20211229

Oilpan: C++ 垃圾回收器



交叉引用
https://research.google/pubs/pub47359/

Oilpan: C++ 垃圾回收器



```
<html>
  <body onload="run()">
    <div>
      <b id="msg">Hello!</b>
    </div>
  </body>
</html>
```

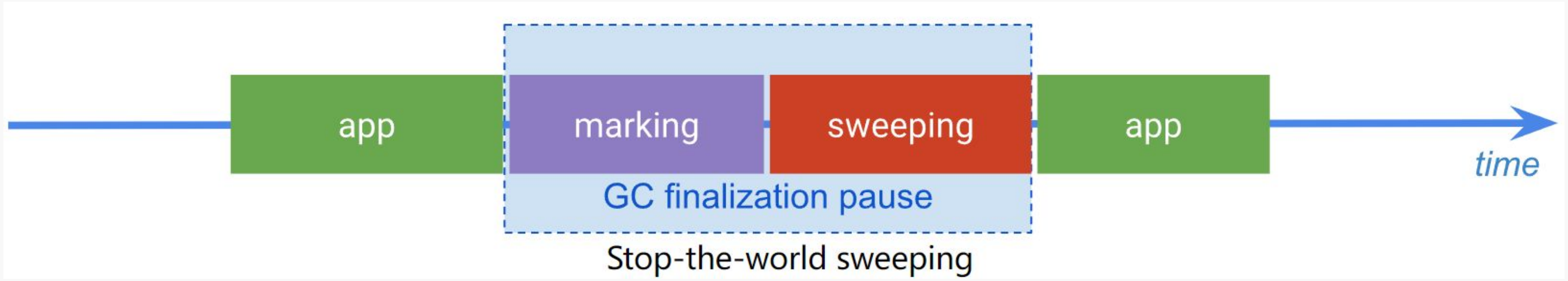
(a) HTML source

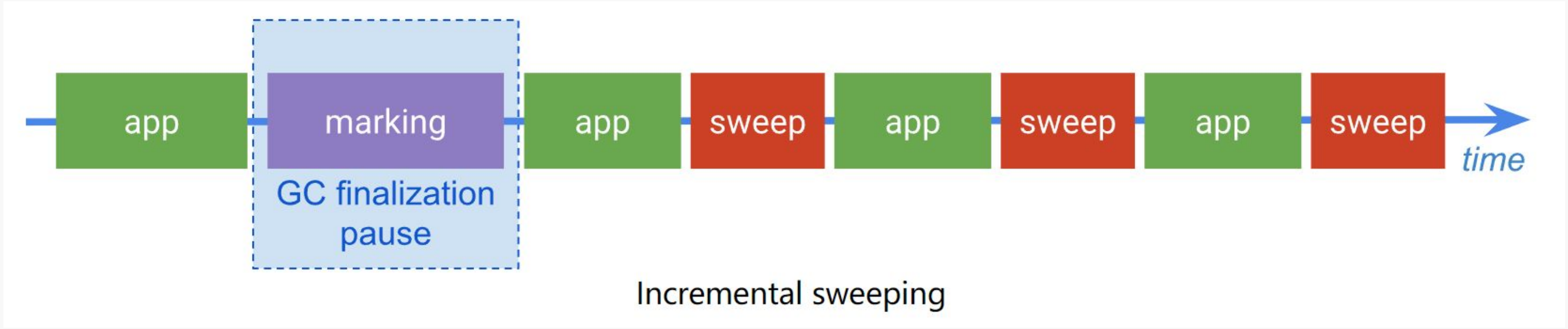
```
class Leak{};
function run() {
  let leak = new Leak();
  function listener() { console.log(leak); }
  let node = document.getElementById("msg");
  node.addEventListener("debug", listener);
}
```

(b) JavaScript source

代码举例

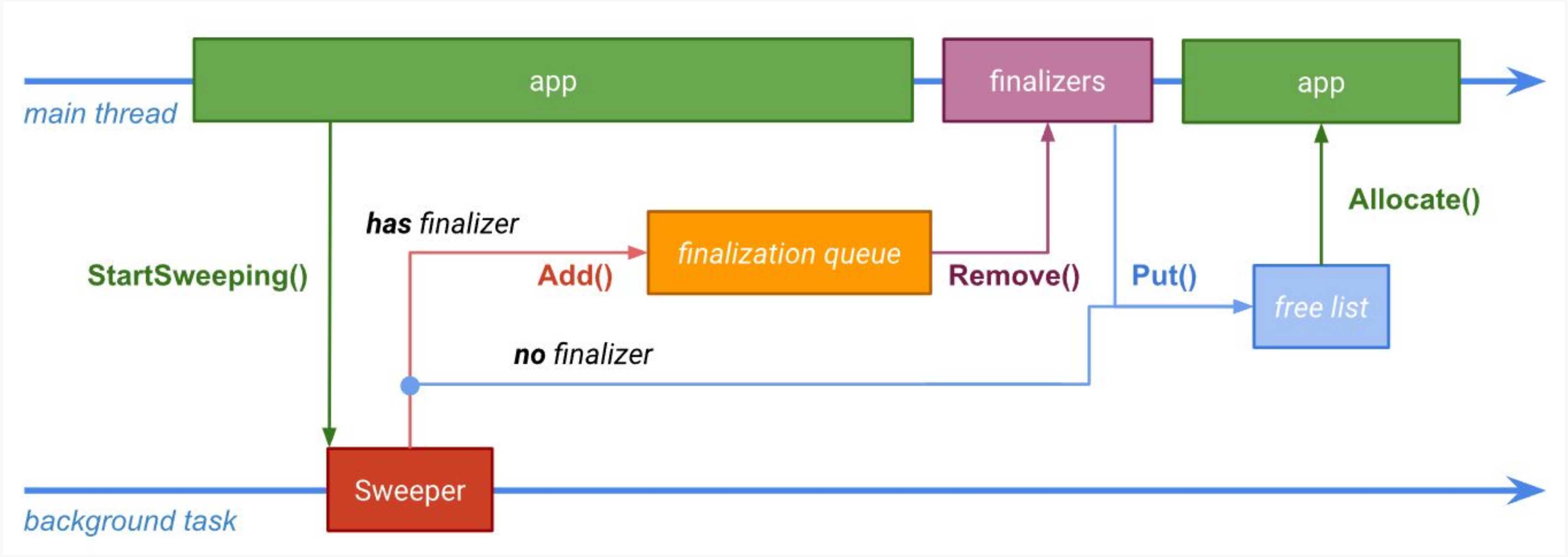
运行时序图





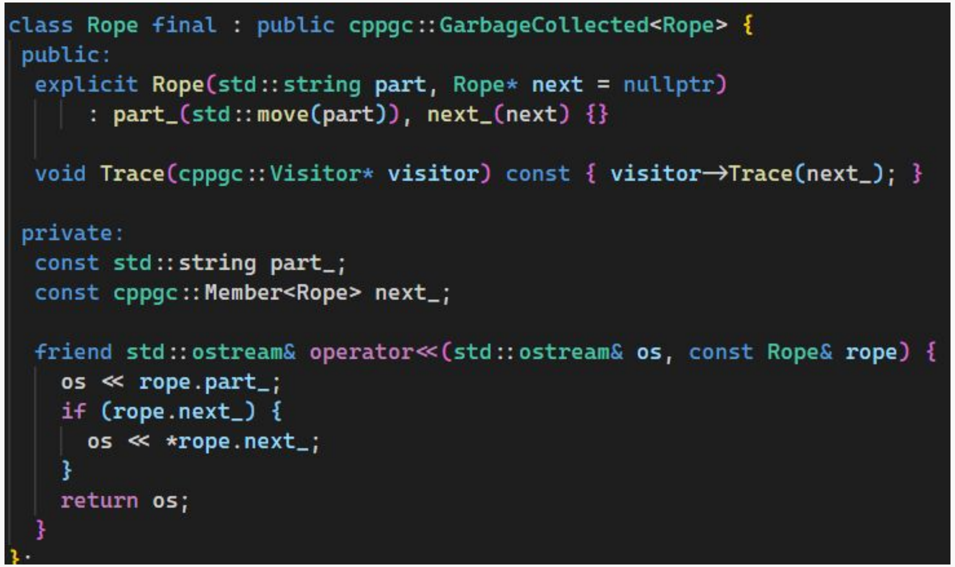
标记回收
https://v8.dev/blog/high-performance-cpp-gc

运行时序图

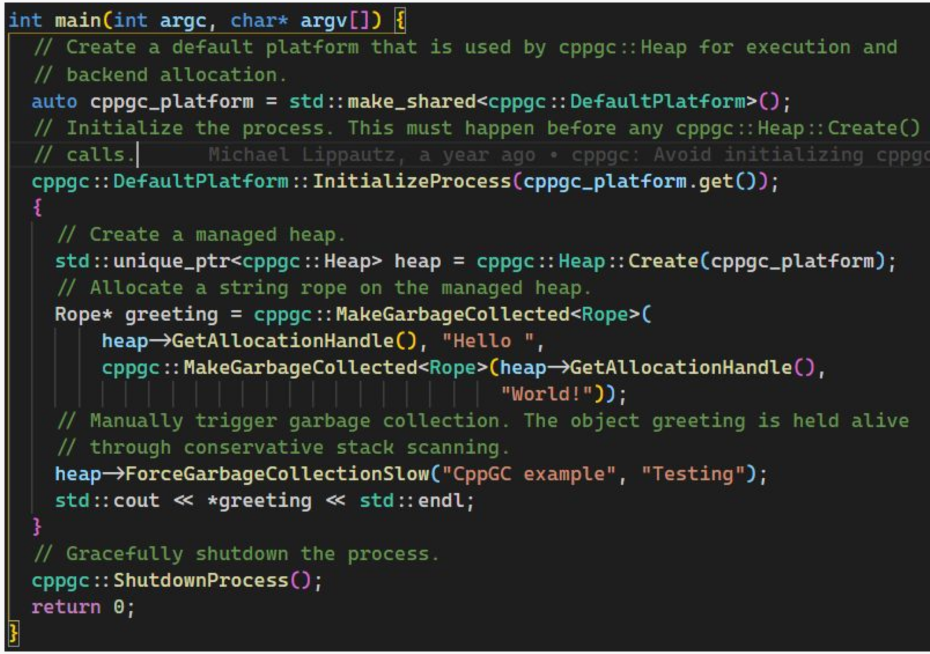


标记回收
https://v8.dev/blog/high-performance-cpp-gc

示例代码



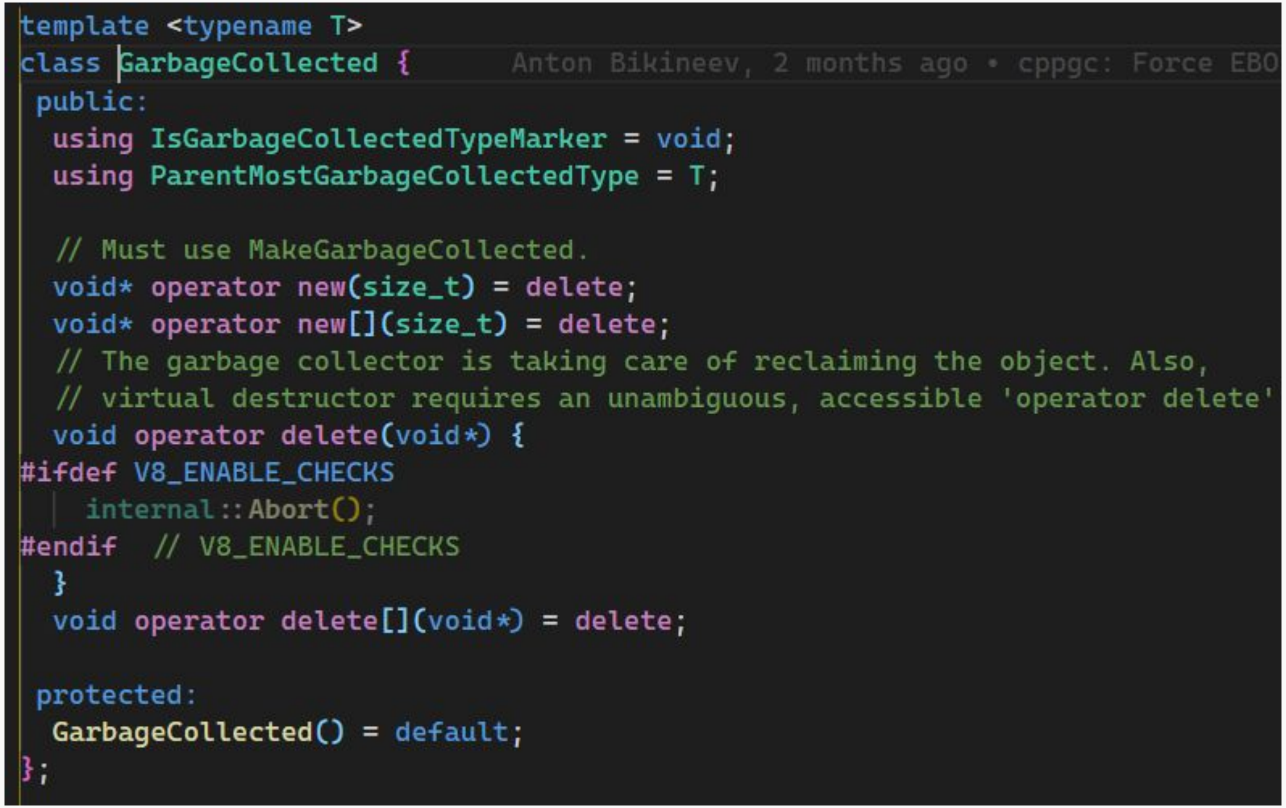
```cpp
class Rope final : public cppgc::GarbageCollected<Rope> {
 public:
  explicit Rope(std::string part, Rope* next = nullptr)
      : part_(std::move(part)), next_(next) {}

  void Trace(cppgc::Visitor* visitor) const { visitor->Trace(next_); }

 private:
  const std::string part_;
  const cppgc::Member<Rope> next_;

  friend std::ostream& operator<<(std::ostream& os, const Rope& rope) {
    os << rope.part_;
    if (rope.next_) {
      os << *rope.next_;
    }
    return os;
  }
};
```



```cpp
int main(int argc, char* argv[]) {
  // Create a default platform that is used by cppgc::Heap for execution and
  // backend allocation.
  auto cppgc_platform = std::make_shared<cppgc::DefaultPlatform>();
  // Initialize the process. This must happen before any cppgc::Heap::Create()
  // calls.       Michael Lippautz, a year ago • cppgc: Avoid initializing cppg
  cppgc::DefaultPlatform::InitializeProcess(cppgc_platform.get());
  {
    // Create a managed heap.
    std::unique_ptr<cppgc::Heap> heap = cppgc::Heap::Create(cppgc_platform);
    // Allocate a string rope on the managed heap.
    Rope* greeting = cppgc::MakeGarbageCollected<Rope>(
        heap->GetAllocationHandle(), "Hello ",
        cppgc::MakeGarbageCollected<Rope>(heap->GetAllocationHandle(),
                                          "World!"));
    // Manually trigger garbage collection. The object greeting is held alive
    // through conservative stack scanning.
    heap->ForceGarbageCollectionSlow("CppGC example", "Testing");
    std::cout << *greeting << std::endl;
  }
  // Gracefully shutdown the process.
  cppgc::ShutdownProcess();
  return 0;
}
```

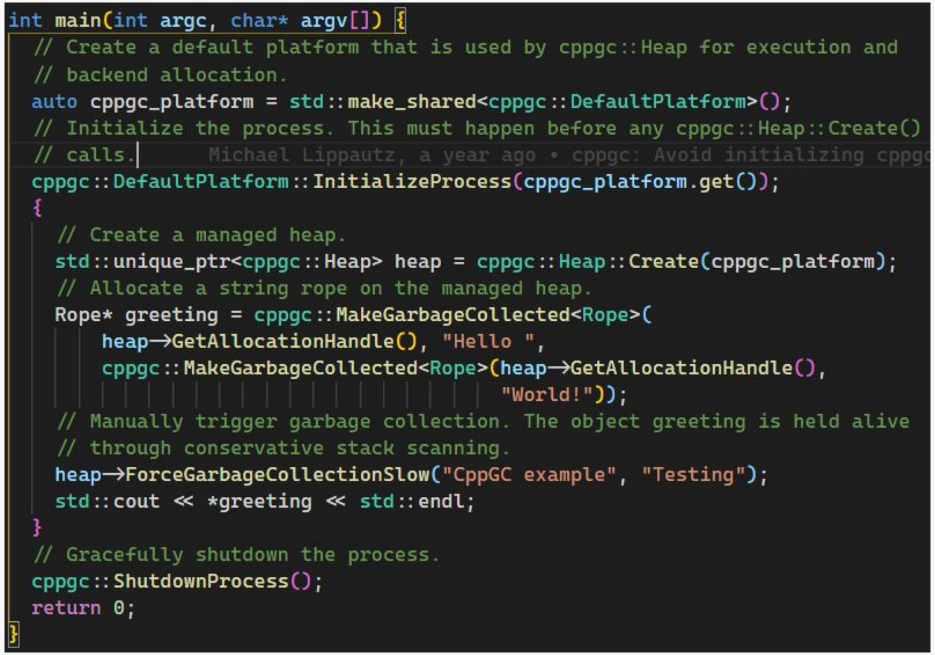
https://source.chromium.org/chromium/chromium/src/+/master:v8/samples/cppgc/hello-world.cc

示例代码



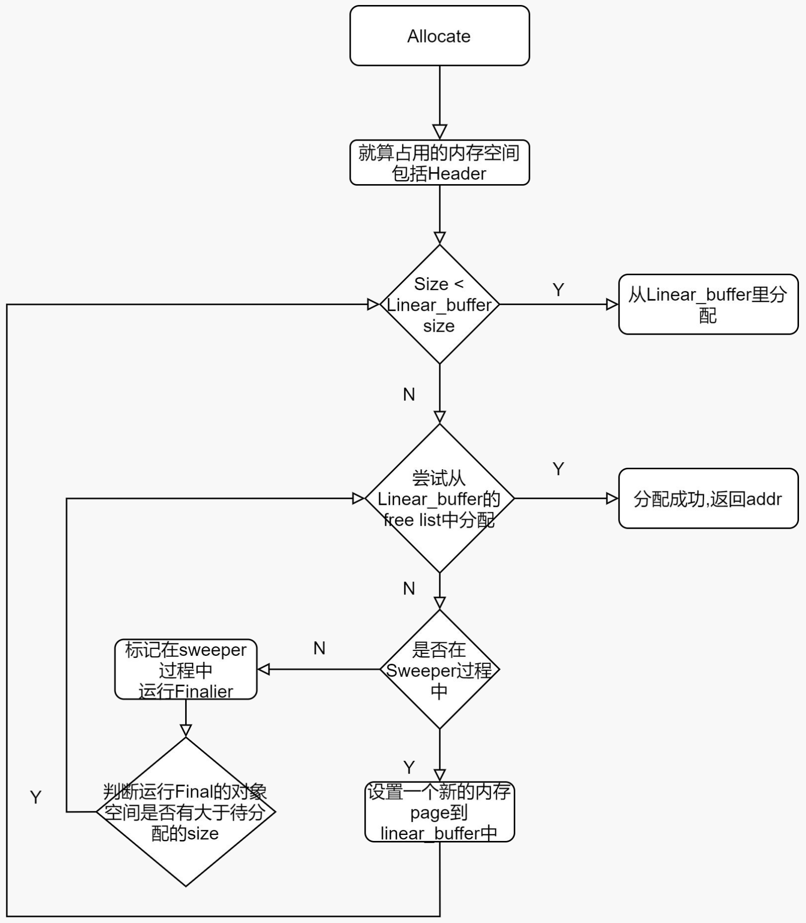
```
template <typename T>
class GarbageCollected {      Anton Bikineev, 2 months ago • cppgc: Force EBO
 public:
  using IsGarbageCollectedTypeMarker = void;
  using ParentMostGarbageCollectedType = T;

  // Must use MakeGarbageCollected.
  void* operator new(size_t) = delete;
  void* operator new[](size_t) = delete;
  // The garbage collector is taking care of reclaiming the object. Also,
  // virtual destructor requires an unambiguous, accessible 'operator delete'
  void operator delete(void*) {
#ifdef V8_ENABLE_CHECKS
    internal::Abort();
#endif  // V8_ENABLE_CHECKS
  }
  void operator delete[](void*) = delete;

 protected:
  GarbageCollected() = default;
};
```

示例代码



```cpp
int main(int argc, char* argv[]) {
  // Create a default platform that is used by cppgc::Heap for execution and
  // backend allocation.
  auto cppgc_platform = std::make_shared<cppgc::DefaultPlatform>();
  // Initialize the process. This must happen before any cppgc::Heap::Create()
  // calls.      Michael Lippautz, a year ago • cppgc: Avoid initializing cppgc
  cppgc::DefaultPlatform::InitializeProcess(cppgc_platform.get());
  {
    // Create a managed heap.
    std::unique_ptr<cppgc::Heap> heap = cppgc::Heap::Create(cppgc_platform);
    // Allocate a string rope on the managed heap.
    Rope* greeting = cppgc::MakeGarbageCollected<Rope>(
        heap->GetAllocationHandle(), "Hello ",
        cppgc::MakeGarbageCollected<Rope>(heap->GetAllocationHandle(),
                                          "World!"));
    // Manually trigger garbage collection. The object greeting is held alive
    // through conservative stack scanning.
    heap->ForceGarbageCollectionSlow("CppGC example", "Testing");
    std::cout << *greeting << std::endl;
  }
  // Gracefully shutdown the process.
  cppgc::ShutdownProcess();
  return 0;
}
```

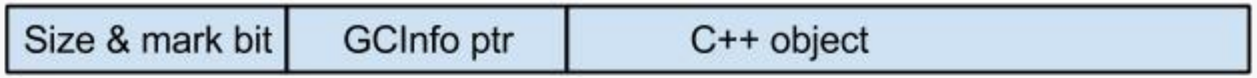
Allocate过程



Allocate
就算占用的内存空间包括Header
Size < Linear_buffer size
Y
从Linear_buffer里分配
N
尝试从Linear_buffer的free list中分配
Y
分配成功,返回addr
N
是否在Sweeper过程中
N
标记在sweeper过程中运行Finalier
判断运行Final的对象空间是否有大于待分配的size
Y
Y
设置一个新的内存page到linear_buffer中

Cppgc对象内存分配流程 图

内存布局



对象的内存布局
https://docs.google.com/document/d/1y7_0ni0E_kxvrah-QtnreMlzCDKN3QP4BN1Aw7eSLfY