Garbage Collection
» 垃圾回收机制 «

- 杨乐 -

- 计 64 -

2017 年 5 月 23 日

Introduction

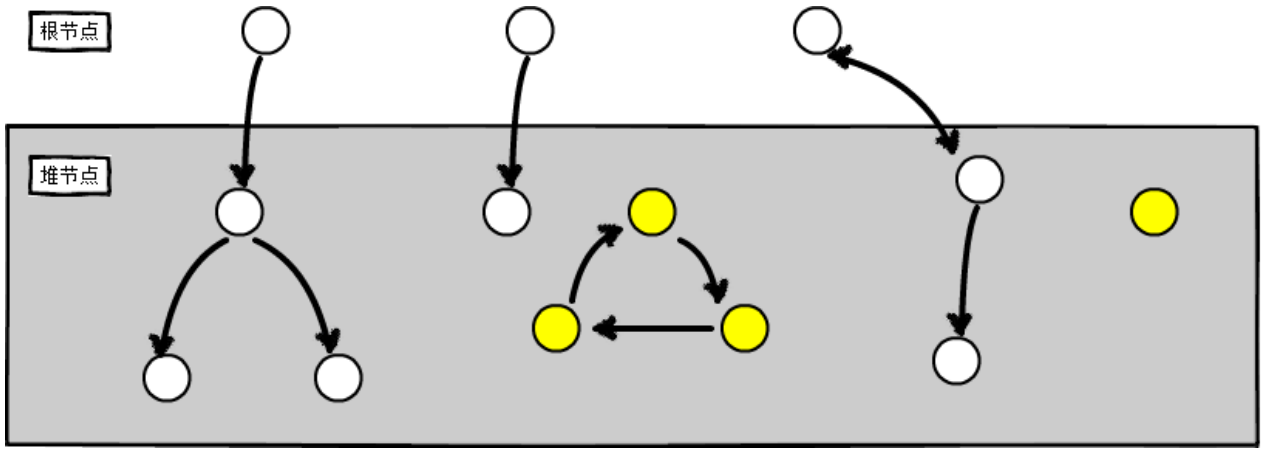
» **Objects: Live or Dead?**

@ **Living Objects**: Objects that are referenced

@ **Dead Objects**: Objects that are no longer referenced

» **Garbage**: All the dead objects

» **Garbage Collection(GC)**: Find and reclaim all the garbage



Introduction

» **GC vs Manual memory managment**:

@ **GC**: Automatically collecting

@ **Manually**: require the programmer to specify which objects to deallocate and return to the memory system

» **Program Languages: GC or Not?**

@ **GC Langs**: Java, C#, D, Go, most scripting languages

@ **Not GC Langs**: C, C++

```
// C++ : Require deallocate manually
int *p = new int(5);
delete p;
```

Introduction

» **Not GC Langs** : C, C++,

- manual memory management, but have garbage-collected implementations available
- Example : *shared_ptr*
- *(GC & mannual : coexist)*

```
// shared_ptr : GC implementations
shared_ptr<int> p( new int(5) );
// **did not need to deallocate manually
```

GC : Advantages & Disadvantages

» Advantages

- dangling pointer bugs (悬挂指针)
- double free bugs (重复释放)
- memory leaks (内存泄漏)

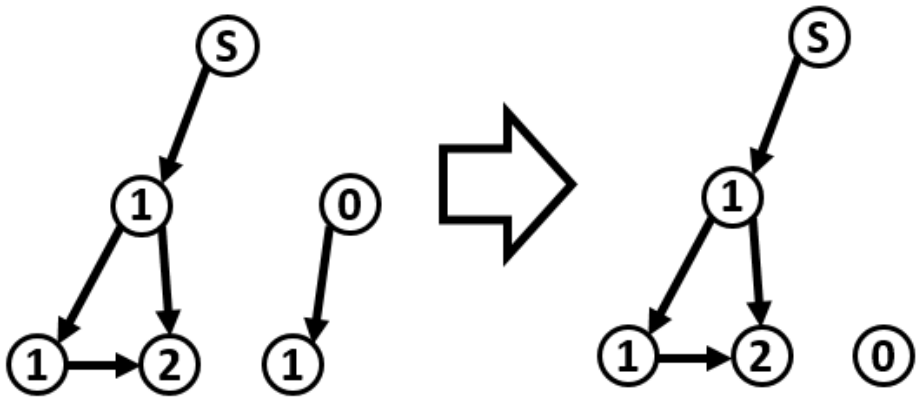
» Disadvantages

- consuming additional resources
- performance impacts (*IOS not adopting GC*)
- incompatibility with manual

GC Strategy: Reference counting

» **Reference Counting** (引用计数法):

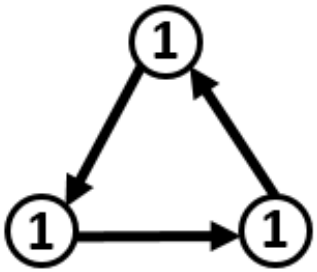
- for each object, a count of the number of references to it held by other objects
- if count reaches zero, -> destroy
  - @ **destroyed / overwritten** -> decrement
  - @ **created / copied** -> increment
- *C++11: shared_ptr*

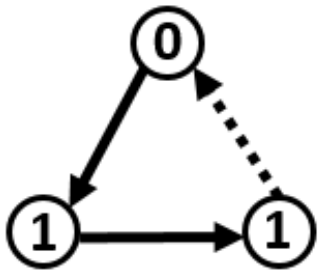


GC Strategy: Weak Reference

» The problem of **Reference Counting** :

- Reference Cycle : count > 0, can not be destroyed
- introduce **Weak Reference**:
  - @ **Strong Reference** : parent-to-child relationships
  - @ **Weak Reference** : child-to-parent relationships
- if **strong reference count** reaches zero, -> destroy
- *C++11: weak_ptr*

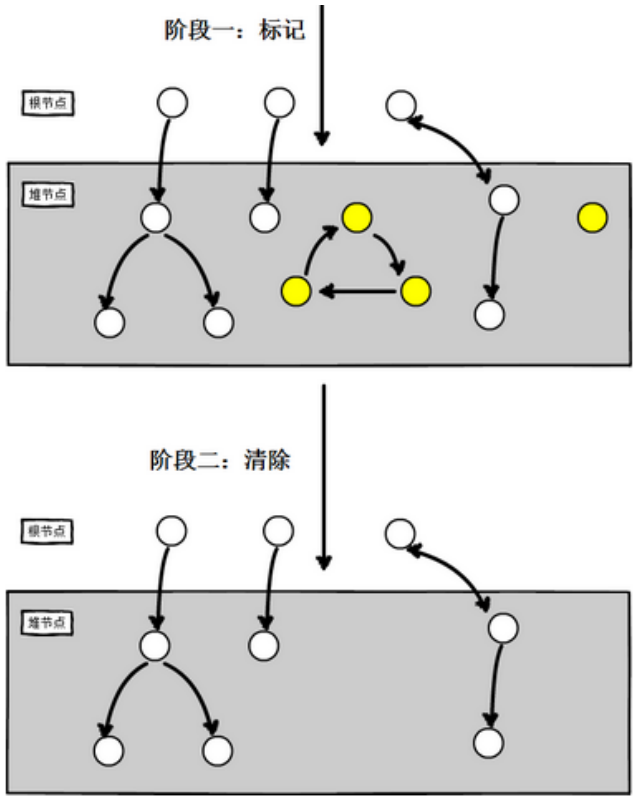




GC Strategy: Root Set

» Mark the reference relations as a **Graph**, and tracing from the root

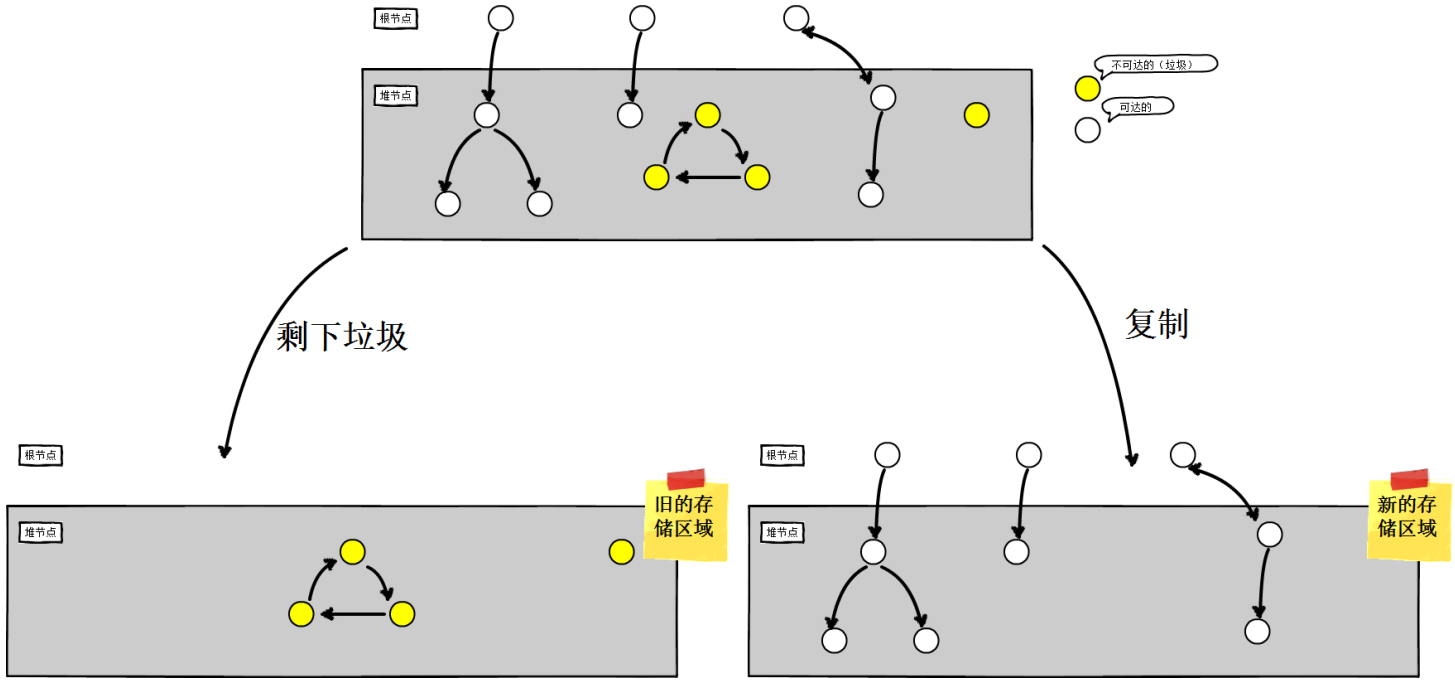
- **Mark - Sweep** : Tracing all the live objects, Collect all the dead



GC Strategy: Root Set

» Mark the reference relations as a **Graph**, and tracing from the root

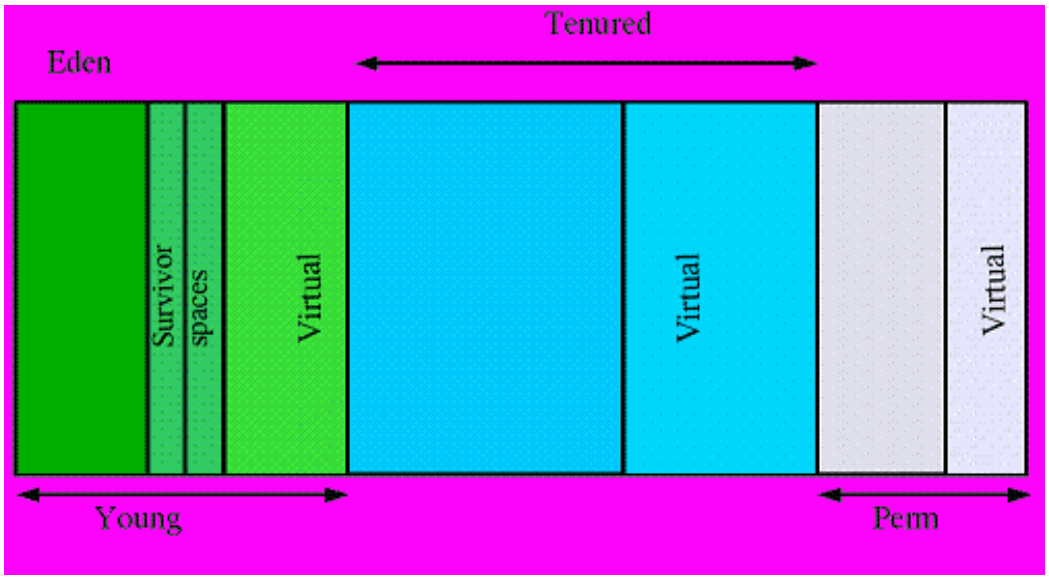
- **Copying** : Copying all the live objects



GC Strategy: Root Set

» Mark the reference relations as a **Graph**, and tracing from the root

- **Generational Collecting** : Divide all the objects into generations



GC Implementation

» **C++(Reference Counting)**:

- *(shared_ptr, weak_ptr, unique_ptr)*

» **Java(Generational Collecting)**:

- *(JVM)*
- *(StrongReference, WeakReference, SoftReference, PhantomReference)*

» **Python(Reference Counting + Generational Collecting)**

- *(Default)*