# 令牌引用机制

### 术语解释：

1. **引用**：本文档主要描述的是对堆内存的管理机制，故本文中的引用主要指代对堆内存的引用，在C语言中可以理解为堆内存指针。
2. **引用主体：**当一个对象内存在引用，引用了其他对象，则该对象就是被引用对象的引用主体。
3. **堆内存对象**：在堆内存中创建的对象，在C++等现代编程语言中可以理解为new关键字所创建的对象本身。
4. **令牌：**一种标识。是对对象可达路径的一种体现，具有传递性。当对象不具有令牌时代表对象已经不能从任何路径访问到，此时对象被回收。

### 原理描述

令牌引用机制可以利用令牌来作为对象的可达性依据。当使用new操作符新建对象A时，对象A内部会维护一张令牌表和一张引用表，最初两张表均为空。

令牌的产生及交换遵循一下几种性质：

**连通性：**只有处于栈内存，静态变量的引用才可以产生令牌。一张令牌就代表一个引用关系的最终出口。

**继承性：**当一个对象被新的指针所指向时，该对象从引用者得到引用者所持有的令牌的副本。

**传递性：**当一个对象获得了一块新的令牌之后，它的所有成员变量所指向的堆内存对象均获得此令牌。若已经持有相同令牌，则不重复持有。

**回收性：**当指向一个对象A的引用被销毁时，若还有引用主体可以提供被销毁的引用所在的引用主体所提供一个或多个令牌时，则对象A继续持有这些令牌。不能提供给对象A的令牌则被回收。当一个对象失去所有令牌时，对象立即被销毁。

上述规则需要实现者建立相应的数据结构为其提供相应的支持，可能需要令牌表，引用表和指针本身的附加行为。在相应的数据结构中，可以采用按内存增长方向进行的排序，从而大幅提高令牌的收发速度。

### 场景示例

引用一般呈图状分布，我们在任意的图结构中任取一个节点，则该节点与其它节点的关系必定为下面三种情况之一：线性，分支，合并。

下面我们对上述三种情况进行讨论，并延申讨论著名的循环引用问题。

1. **线性引用**

对于线性引用，我们以3个对象来作为例子。

A

B

C

Stack

如上图，栈中的局部变量对A的引用是一个入口，故A对象产生一张令牌T。根据令牌的继承性，B C对象分别也持有令牌T. 当对A的引用被销毁时，A失去令牌T，根据令牌的回收性，B C均失去令牌T。故当C失去所有令牌时，C被销毁。接着B回收完成令牌，也不再持有令牌，故被销毁。同理A被销毁。

1. **分支引用**

A

B

C

Stack

对于分支引用，由于对A的引用处于栈中，故A持有一张令牌T。根据令牌的继承性，B C均持有令牌T。销毁对A的引用时，A失去令牌T，从而根据令牌的回收性，B C均失去令牌T，根据上面讨论的过程，B C A依次被销毁。

1. **合并引用**

A

B

Stack

C

对于合并引用，由于对B的引用和对C的引用均处于栈中，故分别持有令牌T1，T2. 故B持有T1，C持有T2令牌。根据令牌的继承性，A持有T1， T2两张令牌。当对B的引用被销毁时，B失去令牌T1，A接着失去T1。由于A还持有T2，故A继续生存。接着B由于失去了所有令牌，立即被释放。接着当指向C的引用被释放时，C和A均失去令牌T2，根据上文讨论的流程，A和C依次被释放。

1. **循环引用**

B

A

Stack

C

2

1

4

3

循环引用其实就是上述线性引用与合并引用的一种组合。但由于循环引用是一个长期困扰内存回收机制的历史问题，在这里作特别讨论。

为了方便说明，我们对上面的引用进行了编号。首先1号引用由于处于出口位置，为A提供了令牌T，根据令牌的继承性，B C依次获得了令牌T。当引用1被销毁时，A失去令牌T，于是顺着引用2，3的路线，B，C依次失去令牌T。这里要特别说明的是，令牌的回收会继续传递到引用4，试图回收A的令牌T。而此时A早已失去了令牌T，故回收过程到此截止。接下来由于C B A均失去了所有令牌，故依次被销毁。