

- [文章](#)
- [站点](#)
- [主题](#)
- [公开课](#)
- [活动](#)
- [客户端](#) 草
- [周刊](#)
  - [编程狂人](#)
  - [设计匠艺](#)
  - [创业周刊](#)
  - [科技周刊](#)
  - [Guru Weekly](#)
  - [一周拾遗](#)

搜索

# 不只是给面试加分 – Java WeakReference的理解与使用

• [登录](#)时间 2014-01-23 07:49:42 [IT社区推荐资讯](#)原文 <http://itindex.net/detail/47754-面试-java-weakreference>主题 [JVM](#)

前言: 看到篇帖子, 国外一个技术面试官在面试senior java developer的时候, 问到一个weak reference相关的问题. 他没有期望有人能够完整解释清楚weak reference是什么, 怎么用, 只是期望有人能够提到这个concept和java的GC相关. 很可惜的是, 20多个拥有5年以上java开发经验的面试者中, 只有两人知道weak reference的存在, 而其中只有一人实际用到过他. 无疑, 在interviewer眼中, 对于weak reference的理解和应用在面试中给了这一个interviewee相当多的加分. 所以, 将我对于这个技术的理解和使用总结在这篇博客里, 希望读者和自己通过读和写这篇帖子, 能够在以后的工作和面试中获得加分.

在Java里, 当一个对象o被创建时, 它被放在Heap里. 当GC运行的时候, 如果发现没有任何引用指向o, o就会被回收以腾出内存空间. 或者换句话说, 一个对象被回收, 必须满足两个条件: 1)没有任何引用指向它 2)GC被运行.

在现实情况写代码的时候, 我们往往通过把所有指向某个对象的referece置空来保证这个对象在下次GC运行的时候被回收 (可以用java -verbose:gc来观察gc的行为)

```
Object c = new Car();
c=null;
```

但是, 手动置空对象对于程序员来说, 是一件繁琐且违背自动回收的理念的. 对于简单的情况, 手动置空是不需要程序员来做的, 因为在java中, 对于简单对象, 当调用它的方法执行完毕后, 指向它的引用会被从stack中popup, 所以他就能在下一次GC执行时被回收了.

但是, 也有特殊例外. 当使用cache的时候, 由于cache的对象正是程序运行需要的, 那么只要程序正在运行, cache中的引用就不会被GC给(或者说, cache中的reference拥有了和主程序一样的life cycle). 那么随着cache中的reference越来越多, GC无法回收的object也越来越多, 无法被自动回收. 当这些object需要被回收时, 回收这些object的任务只有交给程序编写者了. 然而这却违背了GC的本质(自动回收可以回收的objects).

所以, java中引入了weak reference. 相对于前面举例中的strong reference:

```
Object c = new Car(); //只要c还指向car object, car object就不会被回收
```

当一个对象仅仅被weak reference指向, 而没有任何其他strong reference指向的时候, 如果GC运行, 那么这个对象就会被回收. weak reference的语法是:

```
WeakReference<Car> weakCar = new WeakReference(Car)(car);
```

当要获得weak reference引用的object时, 首先需要判断它是否已经被回收:

```
weakCar.get();
```

如果此方法为空, 那么说明weakCar指向的对象已经被回收了.

下面来看一个例子:

```

package weakreference;
/**
 * @author wison
 */
public class Car {
    private double price;
    private String colour;

    public Car(double price, String colour){
        this.price = price;
        this.colour = colour;
    }

    public double getPrice() {
        return price;
    }
    public void setPrice(double price) {
        this.price = price;
    }
    public String getColour() {
        return colour;
    }
    public void setColour(String colour) {
        this.colour = colour;
    }

    public String toString(){
        return colour +"car costs $" +price;
    }
}

```

```

package weakreference;

import java.lang.ref.WeakReference;

/**
 * @author wison
 */
public class TestWeakReference {

    public static void main(String[] args) {

        Car car = new Car(22000,"silver");
        WeakReference<Car> weakCar = new WeakReference<Car>(car);

        int i=0;

        while(true){
            if(weakCar.get()!=null){
                i++;
                System.out.println("Object is alive for "+i+" loops - "+weakCar);
            }else{
                System.out.println("Object has been collected.");
                break;
            }
        }
    }
}

```

在上例中, 程序运行一段时间后, 程序打印出"Object has been collected." 说明, weak reference指向的对象的被回收了.

**值得注意的一点**, 即使有 *car* 引用指向对象, 且 *car* 是一个strong reference, weak reference *weakCar*指向的对象仍然被回收了. 这是因为java的编译器在发现进入while循环之后, *car* 已经没有被使用了, 所以进行了优化(将其置空?). 当把TestWeakReference.java修改为:

```

package weakreference;

import java.lang.ref.WeakReference;

/**
 * @author wison
 */
public class TestWeakReference {

    public static void main(String[] args) {

        Car car = new Car(22000,"silver");
        WeakReference<Car> weakCar = new WeakReference<Car>(car);
    }
}

```

```

int i=0;

while(true){
    System.out.println("here is the strong reference 'car' "+car);
    if(weakCar.get()!=null){
        i++;
        System.out.println("Object is alive for "+i+" loops - "+weakCar);
    }else{
        System.out.println("Object has been collected.");
        break;
    }
}
}

```

weak reference指向的object就不会被回收了. 因为还有一个strong reference car 指向它.

\* WeakReference的一个特点是它何时被回收是不可确定的, 因为这是由GC运行的不确定性所确定的. 所以, 一般用weak reference引用的对象是有价值被cache, 而且很容易被重新被构建, 且很消耗内存的对象.

## ReferenceQueue

在weak reference指向的对象被回收后, weak reference本身其实也就没有用了. java提供了一个ReferenceQueue来保存这些所指向的对象已经被回收的reference. 用法是在定义WeakReference的时候将一个ReferenceQueue的对象作为参数传入构造函数.

## 其他类型的references

-SoftReference

soft reference和weak reference一样, 但被GC回收的时候需要多一个条件: 当系统内存不足时(GC是如何判定系统内存不足? 是否有参数可以配置这个threshold?), soft reference指向的object才会被回收. 正因为有这个特性, soft reference比weak reference更加适合做cache objects的reference. 因为它可以尽可能的retain cached objects, 减少重建他们所需的时间和消耗.

-phantomReference

这个还没想到应用场景, 就先不说了. 有人在实践中用到了的话, 欢迎分享.

## WeakHashMap

to be continued



分享



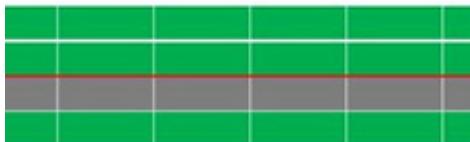
收藏

纠错

推荐文章

- 1. [Spring的事务管理机制](#)
- 2. [Java堆内存](#)
- 3. [跟开涛学SpringMVC（6.3）：Spring MVC 3.1新特性 生产者、消费者请..](#)
- 4. [Java多线程干货系列——（四）volatile关键字](#)
- 5. [MyBatis Generator实现MySQL分页插件](#)
- 6. [详解java定时任务](#)

相关推刊



- 刊主 : zcm 对象 未使用 可回收 [《java》 14](#)



- 刊主 : FredKang [《默认推刊》 1](#)



- 刊主 : 孟莹 [《技术》 2](#)

我来评几句

请输入评论内容 . . .

[登录后评论](#)

已发表评论数(0)

相关站点



[IT社区推荐资讯](#)

+订阅

热门文章

- 1. [Spring的事务管理机制](#)
- 2. [Java堆内存](#)
- 3. [跟开涛学SpringMVC \(6.3\) : Spring MVC 3.1新特性 生产者、消费者请求限定](#)
- 4. [Java多线程干货系列— \(四\) volatile关键字](#)

• 5. [MyBatis Generator实现MySQL分页插件](#)

收藏到推刊

[创建推刊](#)

[收藏](#) [取消](#)

已收藏到推刊！

推刊名(必填)  请填写推刊名

推刊描述

描述不能大于100个字符!

权限设置： 公开  仅自己可见

[创建](#) [取消](#)

x

## 文章纠错

邮箱地址

错误类型  正文不准确 ▾

补充信息

网站相关

[关于我们](#)  
[移动应用](#)  
[建议反馈](#)

[提交](#)

关注我们

 [推酷网](#)

 tuicool2012

友情链接

[人人都是产品经理](#) [PM256](#) [移动信息化](#) [行晓网](#) [Code4App](#) [智城外包网](#) [虎嗅](#) [IT耳朵](#) [艾瑞网](#) [创媒工场](#) [经理人分享](#) [市场部网](#) [砍柴网](#) [CocoaChina](#) [北风网](#) [云智慧](#) [我赢职场](#) [大数据时代](#) [奇笛网](#) [咕噜网](#) [红联linux](#) [Win10之家](#) [鸟哥笔记](#) [爱游戏](#) [投资潮](#) [31会议网](#) [极光推送](#) [Teambition](#) [硅谷网](#) [leangoo](#) [伙伴云表格](#) [ZEALER中国](#) [OpenSNS](#) [小牛学堂](#) [更多链接>>](#)