为什么匿名内部类参数必须为final类型

1）从程序设计语言的理论上:局部内部类(即:定义在方法中的内部类),由于本身就是在方法内部(可出现在形式参数定义处或者方法体处),因而访问方法中的局部变量(形式参数或局部变量)是天经地义的.是很自然的

2）为什么JAVA中要加上一条限制:只能访问final型的局部变量?

3）JAVA语言的编译程序的设计者当然全实现:局部内部类能访问方法中的所有的局部变量(因为:从理论上这是很自然的要求),但是:编译技术是无法实现的或代价极高.

4）困难在何处?到底难在哪儿?
局部变量的生命周期与局部内部类的对象的生命周期的不一致性!

5）设方法f被调用,从而在它的调用栈中生成了变量i,此时产生了一个局部内部类对象inner_object,它访问了该局部变量i .当方法f()运行结束后,局部变量i就已死亡了,不存在了.但:局部内部类对象inner_object还可能 一直存在(只能没有人再引用该对象时,它才会死亡),它不会随着方法f()运行结束死亡.这时:出现了一个"荒唐"结果:局部内部类对象inner_object要访问一个已不存在的局部变量i!

6）如何才能实现?当变量是final时,通过将final局部变量"复制"一份,复制品直接作为局部内部中的数据成员.这样当局部内部类访问局部变量时,其实真正访问的是这个局部变量的"复制品"(即:这个复制品就代表了那个局部变量).因此:当运行栈中的真正的局部变量死亡时,局部内部类对象仍可以访问局部变量(其实访问的是"复制品"),给人的感觉:好像是局部变量的"生命期"延长了.

那么:核心的问题是:怎么才能使得:访问"复制品"与访问真正的原始的局部变量,其语义效果是一样的呢?
当变量是final时,若是基本数据类型,由于其值不变,因而:其复制品与原始的量是一样.语义效果相同.(若:不是final,就无法保证:复制品与原始变量保持一致了,因为:在方法中改的是原始变量,而局部内部类中改的是复制品)

当变量是final时,若是引用类型,由于其引用值不变(即:永远指向同一个对象),因而:其复制品与原始的引用变量一样,永远指向同一个对象(由于是final,从而保证:只能指向这个对象,再不能指向其它对象),达到:局部内部类中访问的复制品与方法代码中访问的原始对象,永远都是同一个即:语义效果是一样的.否则:当方法中改原始变量,而局部内部类中改复制品时,就无法保证:复制品与原始变量保持一致了(因此:它们原本就应该是同一个变量.)

一句话:这个规定是一种无可奈何.也说明:程序设计语言的设计是受到实现技术的限制的.这就是一例. 因为:我就看到不少人都持这种观点:设计与想法是最重要的,实现的技术是无关紧要的,只要你作出设计与规定,都能实现.

现在我们来看,如果我要实现一个在一个方法中匿名调用ABSClass的例子：

```
public static void test(final String s){
    //或final String s = "axman";
  ABSClass c = new ABSClass(){
   public void m(){
     int x = s.hashCode();


     System.out.println(x);


   }
  };
  //其它代码.
 }
```

从代码上看，在一个方法内部定义的内部类的方法访问外部方法内局部变量或方法参数，是非常自然的事，但内部类编译的时候如何获取这个变量，因为内部类除了它的生命周期是在方法内部，其它的方面它就是一个普通类。那么它外面的那个局部变量或方法参数怎么被内部类访问？编译器在实现时实际上是这样的：

```
public static void test(final String s){
   //或final String s = "axman";
```

昵称：lets.run
园龄：4年
粉丝：1
关注：0
+加关注

< 2012年3月 >

日	一	二	三	四	五	六
26	27	28	29	1	2	3
4	5	6	7	8	9	10
11	12	13	14	15	16	17
18	19	20	21	22	23	24
25	26	27	28	29	30	31
1	2	3	4	5	6	7

搜索

找找看

谷歌搜索

常用链接

我的随笔
我的评论
我的参与
最新评论
我的标签
更多链接

随笔分类

java技术(3)
操作系统
设计模式(1)

随笔档案

2012年5月 (2)
2012年3月 (5)

最新评论

1. Re:为什么匿名内部类参数必须为final类型
@地狱门神可能为了获得一些东西，总是要失去一点东西。有些时候想做到尽善尽美比较困难。...
--lets.run
2. Re:为什么匿名内部类参数必须为final类型
但是，这实际上只是一种选择。
像C#的lambda就不要求这个。貌似是因为C#不仅将这些局部变量提升为了闭包类的成员变量，还将外面函数对这个局部变量的引用都改成了对这个类变量的引用。
--地狱门神

阅读排行榜

1. 为什么匿名内部类参数必须为final类型(1655)
2. Java,paint() repaint() paintComponent() update()这些方法的区别是什么?(893)
3. javaee.jar与servlet-api.jar(354)
4. 深入探讨 Java 类加载器(294)
5. 大量数据中找出相同数字的最有效方法(105)

评论排行榜

1. 为什么匿名内部类参数必须为final类型(2)

推荐排行榜

1. 在博客园安家(1)
2. 为什么匿名内部类参数必须为final类型(1)

```
 class OuterClass$1 extends ABSClass{


  private final String s;
  public OuterClass$1(String s){
     this.s = s;
  }
  public void m(){
     int x = s.hashCode();

     System.out.println(x);

  }
 };

 ABSClass c = new OuterClass$1(s);
 //其它代码.
}
```

即外部类的变量被作为构造方法的参数传给了内部类的私有成员.
假如没有final,那么:

```
 public static void test(String s){
    //或String s = "axman";
  ABSClass c = new ABSClass(){
   public void m(){
     s = "other";
   }
  };
  System.out.println(s);
 }
```

就会编译成:

```
  public static void test(String s){
    //或String s = "axman";

  class OuterClass$1 extends ABSClass{


   private String s;
   public OuterClass$1(String s){
      this.s = s;
   }
   public void m(){
     s = "other";



   }
  };

   ABSClass c = new OuterClass$1 (s);

  }
```

内部类的s重新指向"other"并不影响test的参数或外部定义的那个s.同理如果外部的s重新赋值内部类的s也不会跟着改变。
而你看到的

```
 public static void test(String s){
    //或String s = "axman";
  ABSClass c = new ABSClass(){
   public void m(){
     s = "other";
   }
  };
  System.out.println(s);
 }
```

在语法上是一个s,在内部类中被改变了,但结果打印的出来的你认为是同一的s却还是原来的"axman",你能接收这样的结果吗?
所以final从语法上约束了实际上两个不同变量的一致性(表现为同一变量).

转自http://topic.csdn.net/u/20090217/13/f1727a3c-766d-49c2-b68c-d3b5e154cb6e.html 和 http://blog.csdn.net/axman/article/details/1460544

分类: java技术

好文要顶 关注我 收藏该文

lets.run
关注 - 0
粉丝 - 1
+加关注

1 推荐 0 反对

(请您对文章做出评价)

« 上一篇：装饰者模式
» 下一篇：Java,paint() repaint() paintComponent() update()这些方法的区别是什么?

posted @ 2012-03-30 10:54 lets.run 阅读(1655) 评论(2) 编辑 收藏

发表评论

#1楼 2012-03-30 11:42 | 地狱门神

但是，这实际上只是一种选择。
像C#的lambda就不要求这个。貌似是因为C#不仅将这些局部变量提升为了闭包类的成员变量，还将外面函数对这个
局部变量的引用都改成了对这个类变量的引用。

支持(0) 反对(

#2楼[楼主] 2012-03-30 18:57 | lets.run

@ 地狱门神
可能为了获得一些东西，总是要失去一点东西。有些时候想做到尽善尽美比较困难。

支持(0) 反对(

刷新评论 刷新页面 返回顶部

注册用户登录后才能发表评论，请 登录 或 注册，访问网站首页。

最新IT新闻:
· 中国联通盈利警告！一季度净利暴跌85％
· 索尼中国辟谣：日本地震并不影响CMOS产能
· 特斯拉将为其车主提供30天自动驾驶免费试用服务
· MIT开发人工智能 可检测85%的网络攻击
· bot就是你的品牌
» 更多新闻...

最新知识库文章:
· 架构漫谈（九）：理清技术、业务和架构的关系
· 架构漫谈（八）：从架构的角度看如何写好代码
· 架构漫谈（七）：不要空设架构师这个职位，给他实权
· 架构漫谈（六）：软件架构到底是要解决什么问题？
· 架构漫谈（五）：什么是软件
» 更多知识库文章...