Java 7中的 try-with-resource，在没有这个语法糖的情况下的等价实现是什么?
以下面的 demo 为例，这个问题目测99%的人都写不完全正确，不信来战。

0x01 初试牛刀

```
public static void foo() throws Exception {
    try (AutoCloseable c = dummy()) {
        bar();
    }
}

public static void bar() {
    // may throw exception
}
```

我们凭第一感觉来写一下:

```
public static void foo() throws Exception {

    AutoCloseable c = null;
    try {
        c = dummy();
        bar();
    } finally {
        if (c != null) {
            c.close();
        }
    }
}
```

看起来没什么问题，但是仔细想一下，如果bar()抛出了异常e1，c.close()也抛出了异常e2，调用者会收到哪个呢?

我们来回顾一下Java基础，try catch finally部分

```
public static void foo() {
    try {
        throw new RuntimeException("in try");
    } finally {
        throw new RuntimeException("in finally");
    }
}
```

调用foo()函数最终会抛出什么异常呢?

运行一下:

```
Exception in thread "main"
java.lang.RuntimeException: in finally
```

try中抛出的异常，就被finally中抛出的异常淹没掉了。

0x02 suppressed 异常是什么

回到刚刚的问题，如果 bar() 和 c.close()同时抛了异常，那么调用端应该会收到c.close()抛出的异常e2, 往往这并不是我们想要的。那么怎么样抛出 try 中的异常，同时又不丢掉 finally 中的异常呢?

> Java 7 中为 Throwable 类增加的 addSuppressed 方法。当一个异常被抛出的时候，可能有其他异常因为该异常而被抑制住，从而无法正常抛出。这时可以通过addSuppressed 方法把这些被抑制的方法记录下来。被抑制的异常会出现在抛出的异常的堆栈信息中，也可以通过 getSuppressed 方法来获取这些异常。这样做的好处是不会丢失任何异常，方便开发人员进行调试。

有了上述概念，我们进行改写

```
public static void foo() throws Exception {
    AutoCloseable c = null;
    Exception tmpException = null;
    try {
        c = dummy();
        bar();
    } catch (Exception e) {
        tmpException = e;
        throw e;
    } finally {
        if (c != null) {
            if (tmpException != null) {
                try {
                    c.close();
                } catch (Exception e) {

tmpException.addSuppressed(e);
                }
            } else {
                c.close();
            }
        }
    }
}
```

验证我们的想法 javap -c查看字节码:

```
  public static void foo() throws
java.lang.Exception;
    Code:
       0: invokestatic  #2                  //
Method dummy:()Ljava/lang/AutoCloseable;
       3: astore_0
```

```
       4: aconst_null
       5: astore_1

       6: invokestatic  #3                  //
Method bar:()V

       9: aload_0
      10: ifnull        86
      13: aload_1
      14: ifnull        35

      17: aload_0
      18: invokeinterface #4,  1           //
InterfaceMethod java/lang/AutoCloseable.close:()V
      23: goto          86

      26: astore_2
      27: aload_1
      28: aload_2
      29: invokevirtual #6                 //
Method java/lang/Throwable.addSuppressed:
(Ljava/lang/Throwable;)V
      32: goto          86
      35: aload_0
      36: invokeinterface #4,  1           //
InterfaceMethod java/lang/AutoCloseable.close:()V
      41: goto          86

      44: astore_2
      45: aload_2
      46: astore_1
      47: aload_2
      48: athrow
```

```
      49: astore_3
      50: aload_0
      51: ifnull        84
      54: aload_1
      55: ifnull        78

      58: aload_0
      59: invokeinterface #4,  1            //
InterfaceMethod java/lang/AutoCloseable.close:()V
      64: goto          84

      67: astore        4
      69: aload_1
      70: aload         4
      72: invokevirtual #6                  //
Method java/lang/Throwable.addSuppressed:
(Ljava/lang/Throwable;)V
      75: goto          84
      78: aload_0
      79: invokeinterface #4,  1            //
InterfaceMethod java/lang/AutoCloseable.close:()V
      84: aload_3
      85: athrow
      86: return
    Exception table:
       from    to  target type
          17    23    26   Class
java/lang/Throwable
           6     9    44   Class
java/lang/Throwable
           6     9    49   any
          58    64    67   Class
```

```
java/lang/Throwable
           44      50      49     any
```

从字节码的细节可以看到基本跟我们最后的逻辑一致。

0x03 小结

这篇文章我们讲了 try with resource 语句块的底层字节码实现，一起来回顾一下要点：

- 第一，try-with-resource 语法并不是简单的在 finally 里中加入了closable.close()方法，因为 finally 中的 close 方法如果抛出了异常会淹没真正的异常；
- 第二，引入了 suppressed 异常的概念，能抛出真正的异常，且会调用 addSuppressed 附带上 suppressed 的异常。

0x04 思考

留一个作业：我们没有逐行解析 0x02 中的字节码，你能逐行分析一下每条字节码的含义吗?

欢迎你在留言区留言，和我一起讨论。