Hidden Truths in Dead Software Paths

JAN 11TH, 2016

论文下载：http://www.thewhitespace.de/publications/ehmg15-deadpath.pdf

首先我们来看作者都发现了什么样的死代码

Obviously Useless Code

```
252 boolean isInitValueValid(int v) {
253     if ((v < Integer.MIN_VALUE) || (v > 
Integer.MAX_VALUE)) {
254         return false;
255     }
256     return true;
257 }
```

Confused Conjunctions

Confused || conjunctiom

```
1842 if (maxBits > 4 || maxBits < 8) {
1843    maxBits = 8;
1844 }
1845 if (maxBits > 8) {
1846    maxBits = 16;
1847 }
```

注意到这里的死代码是指第一个分支的隐含的else,注意到代码里的两个if语句块如果颠倒了顺序,就会导致maxBits的赋值错误。

Confused && conjunction

```
337 if (ix<0 && ix>=glyphs.length  /length of an array >= 0/)
{
338     throw new IndexOutOfBoundsException("" + ix);
339 }
```

这里的下表范围的检查本应使用|| 但是使用了&&导致检查失败。

Confused Language Semantics

Instanceof and null confusion

```
381 public boolean containsValue(Attribute attribute) {
382   return
383     attribute != null &&
384     attribute instanceof Attribute &&
385     attribute.equals(...(Attribute)attribute...));
386 }
```

作者这里说你这个参数是Attribute了，就不用检查instanceof Attribute了。

New objects are never null

```
288 doc = new CachedDocument(uri);
289 if (doc==null) return null; // better error handling
needed
```

码农刚从c++转型java过来。

Dead Extensibility

```
189 // For now we set owner to null. In the future, it may be
190 // passed as an argument.
191 Window owner = null;
...
198 if (owner instanceof Frame) { ... }
```

Forgotten Constant

有一行代码初始化rub变量为0，然后再140行之后判断rub>0就继续进行，然而这之间rub的值都没有被修改，导致死代码。

Null Confusion

有一些连续多次检查一个变量是不是null的。。

但是还有另外一个。。

```
372 int bands = bandOfsets.length;
373 ... // [bandOfsets is not used by the next 6 lines of
code]
374 if (bandOfsets == null)
375     throw new ArrayIndexOutOfBoundsException("...");
```

这个的问题在于bandOfsets可能一开始是null，然后载372行就会抛出 NullPointerExceptions， 所以374行的检就变成了死代码。

Range Double Checks

```
1095 if (extendableSpaces <= 0) return;
1096 int adjustment = (targetSpan ? currentSpan);
1097 int spaceAddon = (extendableSpaces > 0) ?
1098                  adjustment / extendableSpaces : 0;
```

(这个可能是extendableSpaces⇐0时将spaceAddon置为0有问题，所以后来又在最开始加了判断)

Contradicting range checks in com.sun.-corba.se.impl.orbutil.ORBUtility

```
331 byte jcVersion = jc.javaSerializationVersion();
332 if (jcVersion >= Message.JAVA_ENC_VERSION) {
333     return Message.JAVA ENC VERSION;
334 } else if (jcVersion > Message.CDR_ENC_VERSION) {
335     return jc.javaSerializationVersion();
336 } ...
```

- JAVE_ENC_VERSION 和 CDR_ENC_VERSION分别是1和0。。。这两个if肯定有一个不应该放在这。。。

Type Confusion

```
3825 public Shape createTransformedShape(Shape pSrc) {
3826     if (pSrc == null) { return null; }
3827         return new Path2D.Double(pSrc, this);
3828 }
```

- 有些调用这个函数的地方传入的参数是一个不能被转换成Path2D.Doulbe的Shape的子类，导致抛出异常。

Unsupported Operation Usage

```
29 static CodecFactory extract (org.omg.CORBA.Any a) {
30     return read (a.create_input_stream ());
31 }
32 static CodecFactory read (InputStream istream) {
33     throw new org.omg.CORBA.MARSHAL ();
34 }
```

程序员似乎是想表明这个类不支持read这个操作，但是调用它的代码似乎并不知道，调用之后抛出异常导致a创建的stream没有关闭。

Unexpected Return Value

```
746 if (isUnnecessaryTransform(...)) {
747     conn.getDestination().setTransform(null);
748 }
...
761 static boolean isUnnecessaryTransform(... transform){
762     if(transform == null) return false;
763     ... // [the next four tests also just return false]
764     return false;
765 }
```

isUnnecessaryTrasform错误的返回值导致死代码。。

Approach

```
1 void conditionInFinally(java.io.File f) {
2   boolean completed = false;
3   try {
4       f.canExecute();
5       completed = true;
6   }  finally {
7   if (completed) doSomething();
8
9 }}
```

```
1 Throwable doFail() { throw new Exception(/ message/); }
2
3 Object compute(Object o) {
4   if(o == null)  return doFail(); OR throw doFail();
5   else return o;
6 }
```

- 到这里，文章的一大半就结束了= =
- 后面，作者介绍寻找死代码的方法是:
- 从任意函数的入口点开始，所有参数的初始值认为是任意值
- 开始使用弱化版value set的方式，模拟执行过程，进行中间变量的值域分析:
    - 只确定变量可能的取值集合,如果集合内元素的个数（不是使用value set 表达需要的大小？）的差大于一个阈值就认为这个值是[minInt, maxInt]的
    - 只分析 int 变量

- 然后作者对一些可以自动化取出的false positive 进行了解释
- finally块.
- 一些语法导致的必要的deadcode
- switch 的default
- Assert
- Reflection (不能处理)

Evalution

- JDK，1.8.0_25
- 556个dead code, 去除上一节提到的自动去除方案去除掉的279个(279*81%是finally块的)
- 剩下的结果经过的人工分析如下:

Category	Percentage
Obviously Useless Code	1%
Confused Conjunctions	2%
Confused Language Semantics	3%
Dead Extensibility	9%
Forgotten Constant	4%
Null Confusion	54%
Range Double Checks	11%
Type Confusion	3%
Unexpected Return Value	5%
Unsupported Operation Usage	7%
False Positives	1%

MaxCallChain	MaxIntRange	有效死代码个数	过滤个数	总数	总时间(s)	超时终止次数
1	2	690	1157	1847	10	0
1	4	699	1172	1871	11	0
1	8	701	1180	1881	13	0
1	16	702	1182	1884	21	0
1	32	702	1182	1884	27	0
2	2	1078	1248	2326	25	0
2	4	1090	1269	2359	31	0
2	8	1093	1277	2370	43	0
2	16	1094	1279	2373	73	0
2	32	1094	1279	2373	149	1
3	2	1189	1252	2441	60	0
3	4	1201	1273	2474	78	0
3	8	1204	1281	2485	139	0
3	16	1205	1283	2488	289	1
3	32	1205	1283	2488	894	10
4	2	1224	1252	2476	156	0
4	4	1236	1273	2509	205	1
4	8	1237	1281	2518	438	7
4	16	1237	1283	2520	1259	27
4	32	1233	1283	2516	6117	63
5	2	1225	1252	2477	457	4
5	4	1235	1273	2508	990	7
5	8	1239	1281	2520	1566	20
5	16	1234	1283	2517	5482	80
5	32	1233	1283	2516	39274	143