Chapter 5 Diagnose the exception

The part of system we name it “Exception diagnose system.” It is use to diagnose the cause of exception is in which components. In fact, we could not ensure finding out where the root cause is. However, we could try to find out some components they are suspicion of causing the exception and show the probability that the component cause the exception.

5.1 The work flow of diagnose the exception

At first, we need to find out any side effect statements about the exception and check them in which components. According to this that system computes the points of each component. Figure 5-1 illustrates this part.

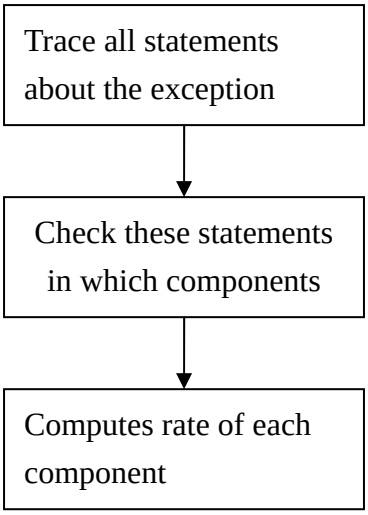


Figure 5-1 : The work flow of diagnose the exception

5.2 Analyze exception using stack trace and slicing profile

The element of stack trace is note the information about the exception. Take an example in Figure 5-2. This is one element of stack trace. We could know the exception was occurred in which class, which method and what line of java source file.

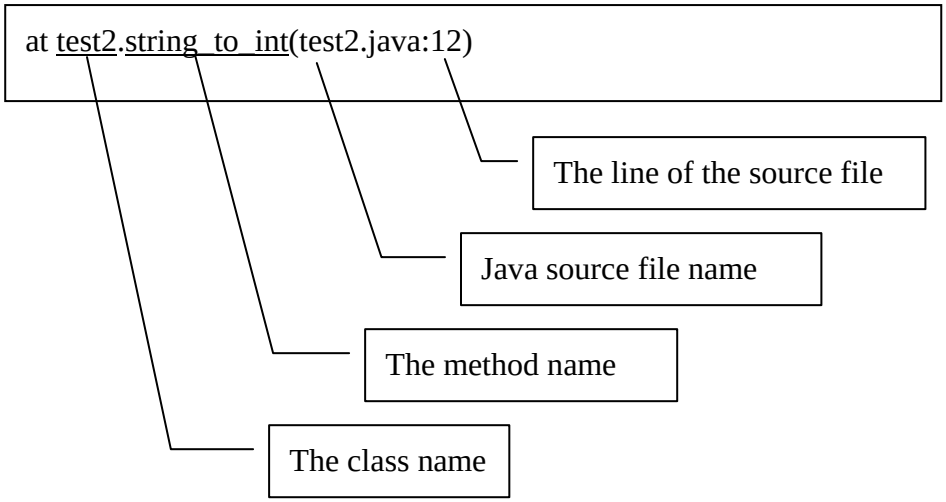


Figure 5-2 : A message string from stack

According to stack trace, we could get the exception point. But some part of stack they are not at the application. They are at the Java virtual machine. We could use the information of the component map to check the class name is in the component or not. If we could not find the class in the component map the we could ensure that the class is in Java virtual machine's lib. For example, in Figure 5-3 the top 3 elements is Java virtual machine's lib. The bottom two is the application. The Figure 5-4 shows the different.

```
at java.lang.NumberFormatException.forInputString(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at test2.string_to_int(test2.java:12)
at test.main(test.java:15)
```

Figure 5-3: An example of stack trace

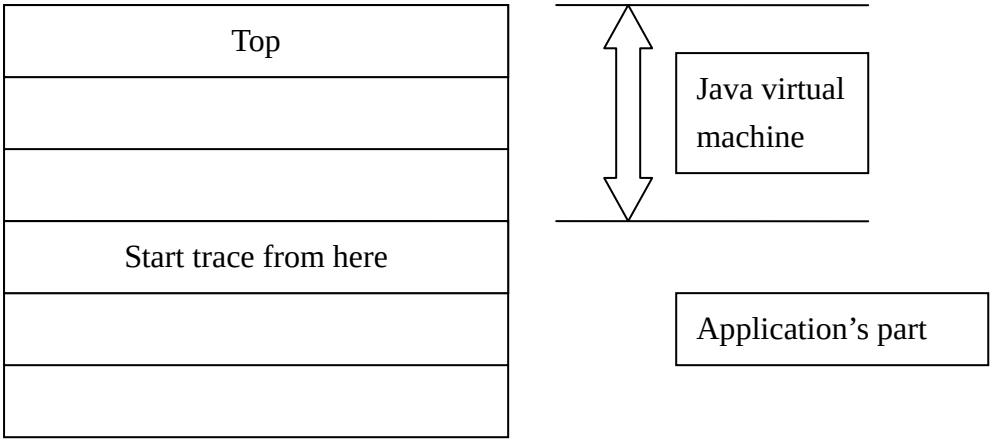


Figure 5-4 : The different part between JVM and application in stack

According to the stack trace, we could know that exception is in what class, method and which line of the source. Take the Figure 5-2 for example that we could know the exception point is in the method *string_to_int* of the class *test2*. Start to diagnose from the line 12 in the test2.java. In the last chap, we have described the information that we prepare for diagnosing exception by using program slicing.

First, pop an element from the stack. We could get the information from the message. We could get statement of the exception is in class *c*, in method *m* and the line number *l*. Second, add *c* to the list that note all class that effect the statement of exception. We could find out what the value *v* in *l* then check the slicing profile of *v*. If there has link to other profile the class of the profile would be established for effecting the statement of the exception. So repeat the second step and add the class of the slicing profile to list. If the *v* is parameter of *m* then pop the next element from the

stack and repeat the first step. The algorithm of these steps is in Figure 5-5.

```
Procedure Traceclass()
begin
   Pop element E from stack
   Get the class c and method m from E
   Get the statement s and the line l
   Get all value in s
      For each value (V)

         Findlink(V, l, line(m))

         If V is parameter then
            Traceclass()
         End if
      Next
end
```

```
Procedure Findlink (profile, s, e)
Begin
   Addclass(getclass(profile))
   For each statement(S) start from s to e
      If S have link then
            Findlink(link(S), line(S), e)
      End if
   Next
End
```

Figure 5-5 : The algorithm of diagnosing exception to trace the stack

5.3 Compute the probability

The probability of a component is computed by the number of its own class be

established that effect the exception in each element of stack. For example, the component *Bean3* have 3 classes be established that effect the exception in element *T*. *Bean3* get the point *Pt*. *Pt = 3 * Tp*. *Tp* is the point of element *T*. The probability of *Bean3* is the sum of points in all elements. The point of an element that we compute is according to the number of element that is traced in stack and the number of class is established that effect the exception in the element. The algorithm is in Figure 5-6.

```
Procedure computeprobability(bean)
Begin
    Probability=0
    For each element E
        Class_num=0
        C=get_class_list(E)
        For each class C
            If C is in bean
                Class_num++;
        Next
        Probability += Class_num * Elementpoint (E)
    Next
End
```

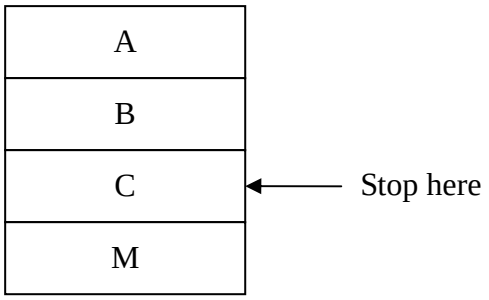
```
Procedure Elementpoint (E)
Begin
  Get the number of element that pop from the stack N
  P=100 / N
  C=get_class_number(E)
  P=P/C
  Return P
End
```

Figure 5-6 : The algorithm of computing the probability of a component

We trace *n* elements in stack. The default point of all elements is *P*. *P* = 100 / *n*. There is *cn* classes that be established in element *E*. The point of element *E* is *Ep*. *Ep* = *P* / *cn*.

Take an example in Figure 5-7, our system traces 3 elements in the stack and there are three components in the application. In the (A) show that it start tracing from

A then stop at the C. In the (B) show the number of effect class that be found in each element. In the (C) show the class that be found that is belong to which component. According to (A), (B), (C), we could compute the probability of each components.



element	Class number
A	1
B	4
C	2
M	0

(A) Start from A. Stop at C

(B) the num of class that be found

Bean	Effect what stack
Bean1	A, B
Bean2	B, B, C
Bean3	C, B

(C) The effect class be found in each bean; the Bean2 have 2 class be found in B

ElementPoint	Ponit
Ap	33/1 = 33
Bp	33/4 = 8.25
Cp	33/2 = 16.5
Mp	0

(D) The point of each element in stack

Component	Probability
Bean1	33 + 8.25 = 41.25
Bean2	8.25 * 2 + 16.5 = 33
Bean3	16.5 + 8.25 = 24.75

(E) Probability of each component

Figure 5-7: An example

First, we need to compute the point of each element. The result is show in the (D) then compute the probability of each component that show in the (E).

5.4 Example

There are several cases they would effect the statement of exception. We use some example of those case to explain how the probability be computed for each components. Follow the steps that told on last section. If we want look for all statements that effect the statement of exception, at first we would get the slice profile of the values or object in the statement of exception and look for any side-effect statements start from the line of the exception backward to the front of method in the slice profile. If a statement is flow statement, it would be link to other slice profile.

5.4.1 The sample case

5.4.1.1 Case 1:

In this case, the exception is in the class *test2*, method "*string_to_int.*" The statement of line 8 in the test2.java only has a value *s*. According to the slice profile of s there is not any side-effect between line 8 (the statement of exception) and line 5 (the front of the method). The probability of the component content the class test2 is 100%.

```
at java.lang.NumberFormatException.forInputString(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at test2.string_to_int(test2.java:8)
at test.main(test.java:15)
```

Figure 5-8 : The stack trace of case1

```
public class test2
{
	String s;
	int count=0;
	public int string_to_int()
	{
		s="xxxxxx";
		return Integer.parseInt(s);
	}
}
```

Figure 5-9 : The source code of case1

5.4.1.2 Case 2:

In the case, we could analyze the other flow statement that effects the statement of exception. There are two components in this application. A component is called *main that* include two classes. They are *test* and *test2*. The other component is called *sub* it include one class test6. The function *is_init* of clss *test6* is effect the statement of exception. The class test6 is suspicious of causing the exception. In our system, the result of diagnosing for this case that *main* is 50% and *sub* is 50%.

```
at java.lang.NumberFormatException.forInputString(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at java.lang.Integer.parseInt(Unknown Source)
at test2.string_to_int(test2.java:13)
at test.main(test.java:15)
```

Figure 5-10 : The stack trace of case2

```
public class test2
{
	String s;
	test6 b=new test6();
	public int string_to_int()
	{
		s="xxxxxx";
		if(!b.is_init())
		{
			return Integer.parseInt(“0”);
		}
		else
			return Integer.parseInt(s);
	}
}
```

Figure 5-11 : The source of case2

5.4.1.3 Case 3:

We use the same component and the same class in this case. In the case, we modify the source code of test.java and test2.java. The source of test.java is in Figure 5-12. The source of test2.java is in Figure 5-13. The stack trace is the same with Figure 5-13. The class test calls the member of test2 *string_to_int* and passes the

parameter. In the test2, we could find the parameter would effect the statement of exception. So, we would trace to the next element of stack. In our system, the result of diagnosing for this case that is *main* 75% and *sub* 25%.

```
public class test
{
	public static void main(String[] args)
	{
		String s="ccc";
		test2 temp=new test2();
		int i=temp.string_to_int(s);
	}
}
```

Figure 5-12: The source of test.java

```
public class test2
{
	test6 b=new test6();
	public int string_to_int(String s)
	{
		if(!b.is_init())
		{
			return Integer.parseInt(“0”);
		}
		else
			return Integer.parseInt(s);
	}
}
```

Figure 5-13: The source code of test2.java