#597494 사는얘기

신입 개발자가 혼자 공부하는 방법 - 스택트레이스로 디버깅하기

비전공자였던 자바개발자가, 혼자 시행착오 겪으며 만든 방법이에요.

웹개발자가 요새 인기가 많죠? 처음 시작하시는 분들에게 도움이 되었으면 좋겠어요.

제일 중요한 세 가지는,

디버깅, 원본소스 보기, 구글링 이였어요.

1. 궁금한 것을 질문하지 않고 혼자 해결할 수 있는 능력
2. 버그를 질문하지 않고 혼자 해결할 수 있는 능력

이 두가지가 부족하면, 결국 질문을 올려야 하거든요.

저도.. 처음엔 혼자 해결못해서 지식인에 질문 700개씩 올리고 했었어요...

또한, 질문은 생각보다 비용이 높아요.

1. 남들이 이해할 수 있도록 질문글을 작성하는데 시간 오래걸리구요.
2. 답변이 달릴때마다 하던 공부를 멈추고 확인하고 또.. 댓글을 달아야 해요.

즉 영양가없는 질문을 올리면, 질문자와 답변자 모두에게 좋지못해요.

서론은 여기까지에요. 목차는 다음과 같아요.

1. 디버깅
(1) 스택트레이스 읽기
(2) 예외클래스 확인하기
(3) 예외가 발생한 메소드 확인하기
(4) Jsp 디버깅 꿀팁

2. 원보소스 살펴보기
(1) 개발하는 언어의 원본 소스 사이트 알아두기
(2) 디컴파일해서 살펴보기

3. 구글링 팁
(1) 모르는 것을 영어문장으로 바꿔보기.
(2) 검색은 공식 사이트부터.
(3) 부작용을 조심하기.

글이 길어져서.... 일부만 적고, 다음글은

https://okky.kr/article/597508

url로 대체할게요.

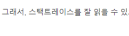
```
public static void main(String[] args)
{
    int[] array = {1, 2};

    array[3] = 5;
```

```
StackTraceTest [Java Application] C:₩Program Files₩Java₩jre1.8.0_202₩bin₩javaw.exe (2019. 7. 2. 오전 7:31:25)
n in thread "main" java.lang.ArrayIndexOutOfBoundsException: 3
at pac1.pac2.StackTraceTest.main(StackTraceTest.java:9)
```

자바의 장점은, 스택트레이스로 버그를 찾기쉽게 해줬다는 거라고 생각해요.

그래서, 스택트레이스를 잘 읽을 수 있으면, 혼자 버그를 질문없이 해결할 수 있어요.



path가, 만약 이렇게 된다면

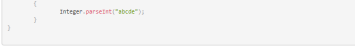
```
package pac1.pac2;

public class StackTraceTest
{
        public static void main(String[] args)
        {
                one();
        }

        public static void one()
        {
                two();
        }

        public static void two()
        {
                three();
        }

        public static void three()
        {
                Integer.parseInt("abcde");
        }
}
```



```
Exception in thread "main" java.lang.NumberFormatException: For input string: "abcde"
	at java.lang.NumberFormatException.forInputString(Unknown Source)
	at java.lang.Integer.parseInt(Unknown Source)
	at java.lang.Integer.parseInt(Unknown Source)
	at pac1.pac2.StackTraceTest.three(StackTraceTest.java:22)
	at pac1.pac2.StackTraceTest.two(StackTraceTest.java:17)
	at pac1.pac2.StackTraceTest.one(StackTraceTest.java:12)
	at pac1.pac2.StackTraceTest.main(StackTraceTest.java:7)
```

이렇게 스택트레이스가 찍혀요.

Exception in thread "쓰레드이름" 예외클래스경로.예외클래스이름: 예외 메시지

at 예외발생한클래스경로.클래스이름.메소드이름(클래스이름:예외발생한라인번호)

이렇게 구성되어있는데, 하나씩 살펴보면 다 이해가 되실거에요.

또한, 잘 살펴보시면

at ... three()

at ... two()

at ...one()

at ...main() 이라고 되어있죠?

이것은 메소드를 호출한 순서에 따라 스택트레이스가 찍힌다는 뜻이에요.

main()에서 one()호출하고

one()에서 two()호출하고

two()에서 three()를 호출하니까요.

일단 저는 아래와같은 순서로 디버깅을 합니다.

Exception in thread "main" java.lang.NumberFormatException: For input string: "abcde"

at java.lang.NumberFormatException.forInputString(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source)

at pac1.pac2.StackTraceTest.three(StackTraceTest.java:22)

at pac1.pac2.StackTraceTest.two(StackTraceTest.java:17)

at pac1.pac2.StackTraceTest.one(StackTraceTest.java:12)

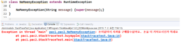
at pac1.pac2.StackTraceTest.main(StackTraceTest.java:7)

읽는 순서는, 노란색 --> 주황색 순서로 읽어요.

노란색)

예외메시지 --> 예외클래스 순서로 읽어요.

스택트레이스에서 제일중요한건 **예외메시지**라고 생각하니까요.



```
public class StackTraceTest
{
    public static void main(String[] args)
    {
        int money = 0;

        buyApple(money);
    }

    public static void buyApple(int money)
    {
        if(money == 0)
            throw new NoMoenyException("돈이없어서 사과를 구매할수없어요. 돈을 더 가져오시던가 해세요");
    }
}

class NoMoenyException extends RuntimeException
{
    NoMoenyException(String message) {super(message);}
}
```

```
Exception in thread "main" pac1.pac2.NoMoenyException: 돈이없어서 사과를 구매할수없어요. 돈을 더 가져오시던가 해세요
	at pac1.pac2.StackTraceTest.buyApple(StackTraceTest.java:15)
	at pac1.pac2.StackTraceTest.main(StackTraceTest.java:9)
```

왜냐하면, 다른사람이 클래스 만들 때 예외를 쉽게 찾을 수 있도록

예외메시지에 최대한 에러상황를 설명해줄때가 많기 때문이에요.

그래서, 예외메시지만 보고도 해결이 되버려서

이후 남아있는 디버깅과정을 스킵하는 경우도 잦아요.

\>>

Exception in thread "main" java.lang.NumberFormatException: For input string: "abcde"

였죠?

익숙하신 분들은 아.. parseInt("string")나 parseDouble("strubg")같은 메소드를 사용했는데

문자열에 숫자가 아닌 문자가 들어가서 발생했구나.. 를 바로 캐치하셔서 바로 해결하실거에요.

하지만 이 에러를 처음겪는 분이라면 이게 뭔소린지 모르기 때문에,

다음 과정를 거치셔야해요.

예외메시지 --> **예외클래스**

java.lang.NumberFormatException << 이게 무슨 클래스인지를 이제 찾아봐야해요.

어떤 경우에 이 예외가 발생하는지를요.

아까 NoMoneyException을 제가 정의할 때 처럼,

개발자가 예외클래스를 만드는 이유가 다 있기 때문이에요.

java 8 NumberFormatException

검색결과 약 352,000개 (0.47초)

NumberFormatException (Java Platform SE 8) - Oracle Docs
https://docs.oracle.com/javase/8/.../java/.../NumberFormatExcepti... ▾ 이 페이지 번역하기
Constructs a NumberFormatException with no detail message. ... Methods inherited from class java.lang.Throwable ... public NumberFormatException(String s).

패키지가 java.lang이죠? 이건 자바에서 기본적으로 제공되는 패키지라서,

본인이 사용하는 jdk버전를 붙여서 검색하시면 되요.

java 8 NumberFormatException 이렇게요.

Class NumberFormatException

java.lang.Object
    java.lang.Throwable
        java.lang.Exception
            java.lang.RuntimeException
                java.lang.IllegalArgumentException
                    java.lang.NumberFormatException

All Implemented Interfaces:
Serializable

public class NumberFormatException
extends IllegalArgumentException

Thrown to indicate that the application has attempted to convert a string to one of the numeric types, but that the string does not have the appropriate format.

그럼 이렇게 이 예외는 어떤상황에서 발생하는지가 나와요.

구글번역기 돌리면,

응용 프로그램에서 문자열을 숫자 형식 중 하나로 변환하려고 시도했지만 문자열에 적절한 형식이없는 것을 나타 내기 위해 throw됩니다.

라고 하네요.

음~ 문장이 좀 어색하긴해도 이 예외는 문자열을 숫자로 변환하려다 문제가 생기면 반환하나봐요.

Exception in thread "main" org.springframework.beans.factory.BeanDefinitionStoreException: IOException parsing XML document from class path resource [abcde];

만약 java에서 기본적으로 제공되는게 아닌,

다른 라이브러리를 사용하다 이런 에러가 발생했다면,

해당 라이브러리의 버전을 붙여서 검색하시면 되요.

위의경우 저는 spring 5.1.7 BeanDefinitionStoreException이라고 검색해서

공식문서를 찾을 수 있었어요.

spring 5.1.7 BeanDefinitionStoreException

검색결과 약 24,300개 (0.49초)

도움말: 한국어 검색결과만 검색합니다. 환경설정에서 검색 언어를 지정할 수 있습니다.

BeanDefinitionStoreException (Spring Framework 5.1.7.RELEASE API)
https://docs.spring.io/spring.../5.1.7.../index.html?.../springframework/.../BeanDefinition...
public class BeanDefinitionStoreException extends FatalBeanException. Exception thrown when a BeanFactory encounters an invalid bean definition: e.g. in ...

아무튼... 그다음에 체크해야하는건

at java.lang.NumberFormatException.forInputString(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source)

at java.lang.Integer.parseInt(Unknown Source)

at pac1.pac2.StackTraceTest.three(StackTraceTest.java:22)

at pac1.pac2.StackTraceTest.two(StackTraceTest.java:17)

at pac1.pac2.StackTraceTest.one(StackTraceTest.java:12)

at pac1.pac2.StackTraceTest.main(StackTraceTest.java:7)

이거에요.

메소드가 호출된 과정이 전부 적혀있는데, 맨 왼쪽에

해당 메소드의 클래스의 경로가 모두 적혀있기 때문에,

2가지를 구분하셔야해요.

1. 남이 만든 클래스인지
2. 제가 만든 클래스인지요.

Integer 이런 클래스는 똑똑한 사람이 만들었기 때문에,

당연히 버그가 있다면 제가 만든 클래스에서 발생했다고 추측하는게 옳기 때문에,

제가 만든 클래스에서 가장 마지막에 명시된 예외라인이 어디인지를 찾는게 합당해요.

그러므로..

at java.lang.NumberFormatException.forInputString(Unknown Source)

패키지가 java로 시작하네요?

제가 만든 패키지는 아까 pac1 이런거였으니까, 이런건 더이상 안읽어도 되요.

at pac1.pac2.StackTraceTest.three(StackTraceTest.java:22)

at pac1.pac2.StackTraceTest.two(StackTraceTest.java:17)

at pac1.pac2.StackTraceTest.one(StackTraceTest.java:12)

at pac1.pac2.StackTraceTest.main(StackTraceTest.java:7)

제가 만든 패키지중, 가장 위에꺼를 보시면 되요.

가장 위에꺼가 가장 나중에 호출된거니까요.

그리고 해당 코드를 가면

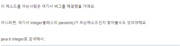
Integer.parseInt("abcde");

이 라인이 있어요. 여기서 예외가 발생했어요.

이 메소드를 아는사람은 여기서 버그를 해결했을 거에요.

아니라면, 여기서 Integer클래스의 parseInt()가 무슨메소드인지 찾아볼수도 있어야해요.

java 8 Integer로 검색해서,



parseInt

```
public static int parseInt(String s)
                    throws NumberFormatException
```

Parses the string argument as a signed decimal integer. The characters in the string must all
to indicate a positive value. The resulting integer value is returned, exactly as if the argumen

Parameters:
s - a String containing the int representation to be parsed
Returns:
the integer value represented by the argument in decimal.
Throws:
NumberFormatException - if the string does not contain a parsable integer.

이 메소드가 무슨 기능을 하는지, 언제 예외를 throw하는지까지 알 수 있어요.

(저처럼 영어못하시면 번역기 돌리시면되요)

이렇게 디버깅하시는게 쌓이시면,

1. 사용하시는 메소드를 좀더 정확하게 사용하여 예외를 예방 or 처리할 수 있게 되구요,
2. 동일한 or 비슷한 예외메시지가 발생하면 예외 메시지만 읽고도 디버깅이 가능해져요.

이렇게 안하고 스택트레이스만 복사붙여넣기 해서 질문글을 올리면,

똑같은 예외를 다음에 만나도 또 질문을 올릴수밖에 없게되요.

https://okky.kr/article/338405

이 글도 추천해드릴게요.

Frudy 2k
2019-07-02 09:04:18 작성 · 2019-07-02 09:07:21 수정됨

2 1624

#597508 사는얘기

신입 개발자가 혼자 공부하는 방법 2편

이 글은,

https://okky.kr/article/597494과 이어지는 글이에요.

1. 디버깅

(1) 스택트레이스 읽기

(2) 예외클래스 확인하기

(3) 예외가 발생한 메소드 확인하기

(4) Jsp 디버깅 꿀팁

2. 원보소스 살펴보기

(1) 개발하는 언어의 원본 소스 사이트 알아두기

(2) 디컴파일해서 살펴보기

3. 구글링 팁

(1) 모르는 것을 영어문장으로 바꿔보기.

(2) 검색은 공식 사이트부터.

(3) 부작용을 조심하기.

——

1. 디버깅

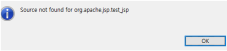
(4) Jsp 디버깅 꿀팁

jsp 디버깅 하다보면 답답한게있어요.

스택트레이스에,

at org.apache.jsp.test_jsp._jspService(test_jsp.java:112)

에러라인이 이렇게 잡혀있어서 저기를 클릭해보면



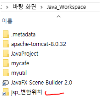
Source not found for org.apache.jsp.test_jsp

이렇게 뜨거든요.

그래서 어쩔수없이 jsp파일이 java로 변환된 위치를 찾아야 하셨죠?

.metadata\.plugins\org.eclipse.wst.server.core\tmp0\wtpwebapps

이런 경로를 매번 외우기 힘드신 분들은,



이렇게 바로가기 파일하나 미리 만들어두면. 앞으로 편하실거에요.

2. 원보소스 살펴보기

(1) 개발하는 언어의 원본 소스 사이트 알아두기

http://hg.openjdk.java.net/jdk8/jdk8/jdk/file/687fd7c7986d/src/share/classes

이건 jdk8버전에서 java의 실제 소스가 들어있는 url이에요.

자바에서 기본적으로 제공되는 클래스를 사용할 때,

그 클래스에 대해 의문이 생기면 원본소소를 살펴보고 해결할 수 있는 경우가 많아요.

저같은경우..

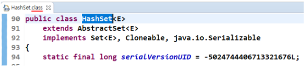
1. https://okky.kr/article/596050

2. https://okky.kr/article/588001

총 두번밖에 못해봤어요.

2. 원보소스 살펴보기

(2) 디컴파일해서 살펴보기



```java
public class HashSet<E>
    extends AbstractSet<E>
    implements Set<E>, Cloneable, java.io.Serializable
{
    static final long serialVersionUID = -5024744406713321676L;
```

위 경로로 찾는게 번거로울때는, 디컴파일해서 살펴볼 수도 있어요.

https://dololak.tistory.com/504

>> 이클립스 쓰시는분들은 이런곳에서 디컴파일러를 보고 설치하시면되요.

다만 주의할점은, 컴파일된 .class 파일을 역으로 디컴파일 하는것이기 때문에,

실제 원본소스와 다를때가 가끔 있어요.

이런게 숙달되면,

https://m.blog.naver.com/PostView.nhn?blogId=tmondev&logNo=220393974518&proxyReferer=https%3A%2F%2Fwww.google.com%2F

이분처럼 디버깅할 수도 있다고 생각해요. 저도 그러고싶어요.

3. 구글링 팁

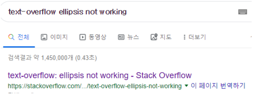
(1) 모르는 것을 영어문장으로 바꿔보기.

제가 모르는 것을 한국어로 표현한다음 영어문장으로 바꿔보는게 좋은거같아요.

예시)

css에서 text-overflow ellipsis; 속성을 사용했는데 이게 안먹히네?

--> text-overflow ellipsis 작동안됨.



text-overflow ellipsis not working

검색결과 약 1,450,000개 (0.43초)

text-overflow: ellipsis not working - Stack Overflow

https://stackoverflow.com/.../text-overflow-ellipsis-not-working ▾ 이 페이지 번역하기

이런식으로 검색하실 수도 있구요,

키워드만으로도 검색하실수도 있어요.

java의 replace(), replaceAll()의 차이를 모르겠다면

replace replaceall difference 이렇게요.

3. 구글링 팁

(2) 검색은 공식 사이트부터.

예를들어, String의 equals()메소드를 검색한다면,

반드시 https://docs.oracle.com/javase/8/docs/api/java/lang/String.html

이렇게 오라클같은 공식문서에 나와있는 equals()설명부터 살펴보셔야해요.

왜냐하면, 구글링에는 생각보다 잘못된 정보가 많아서...

헛수고하실 가능성이 꽤 높아요.

OKKY사이트 같은대에 올라왔다면 신빙성이 그나마 있어요.

잘못된 내용이 있다면 답변에 정정하는 답변이 달렸을거니까요.

하지만 단순 개인블로그라면.... 글쎄요..

3. 구글링 팁

(3) 부작용을 조심하기.

공식사이트 부터 뒤져봐야하는 이유중에 하나이기도 한대요,

어떤 버그가 나서 구글링하면 단순히 임시방편(?)같은 해결책을 줄때가 많아요.

https://okky.kr/article/311337

fender님도 이에 대해 말씀하신적이 있어요.

대충 비유를한다면,

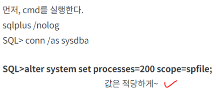
코감기에 걸렸어요 --> 코를 없애세요.

음..비유가 좀 심한거같아요. 아무튼 저런느낌이에요.

```
Caused by: org.springframework.jdbc.CannotGetJdbcConnectionException: Failed to obtain JDBC Connection;
ORA-12519, TNS:no appropriate service handler found
```

저도 이런 에러가 발생해서 결국 구글링 해보니..

대체로 해결방법이...



먼저, cmd를 실행한다.

```
sqlplus /nolog
SQL> conn /as sysdba

SQL>alter system set processes=200 scope=spfile;
```

값은 적당하게~

이렇게 하라네요.

컴퓨터의성능에 따라 적당한 processes값을 계산하는 방법을 알려준것도 아니에요.

그냥 그 값을 사용자가 임의로 지정하라는 듯한 게시글이였어요.

제 입장에서는 상당히 난감하죠.

관련 지식이 없어서, 저 값을 많이 늘렸을 때 생길 부작용에 대해 모르니....

그래서 저는 고민끝에 커넥션풀을 사용해서 해결했었어요.

하지만 구글링에서는 커넥션풀 쓰라는 이야기가 하나도 없었구요.

저도 초급개발자다 보니, 이 부분에 대해서는 더 드릴말씀이 없어요.

그냥 부작용이 있으니 조심하세요! 정도밖에 말씀을 못드릴거같아요.

fender 13k
2016-07-18 20:46:55 작성 · 2018-11-27 07:39:53 수정됨

35 24217

#338405 </> Tips & 강좌 초보 디버깅 예외 스택트레이스

초보 개발자를 위한 스택트레이스 읽는 법

67

70

> *몇 년 전에 네이버 카페에 썼던 글인데, 답변을 달려다 보니 링크가 안되서 이 곳에 옮겨 적습니다.*
> *초보 개발자분들이 제대로 된 디버그 방법을 배우지 못해 오류가 나면 무턱대고 검색부터 하거나 메시지를 통째로 복사해서 질문글만 올리고 어찌할 바를 모르는 모습을 보면 안타까운 생각이 들었는데, 조금이라도 그런 잘못된 습관을 고치는 데 도움이 되었으면 하는 생각입니다.*

면접을 볼 때, 질문할 내용을 미리 정해놓지는 않지만 개인적으로 즐겨내는 문제가 있습니다. 널포인터 예외(*NullPointerException*)의 원인을 찾는 문제인데, 전에 다니던 회사에서는 실제 제품개발 중에 발생한 오류 보고서가 있어서 면접보시는 분들께 해당되는 소스 파일과 함께 제시하고 원인을 찾아보도록 한 적이 있습니다.

처음보는 다른 사람이 짠 소스를 어떻게 분석할 수 있을까 의아할 수 있지만, 실제로 널포인터 예외는 개발 중에 매우 흔히 경험하는 오류이고 특성상 정확한 의미만 알고 있다면 거의 대부분 즉시 정확한 원인을 분석할 수 있습니다. 또한 그러기 위해서는 스택트레이스를 읽을 수 있는 능력이 필요합니다.

개인적으로 예외 처리는 초보 개발자가 가장 중요성을 간과하기 쉬운 분야라고 생각합니다. 예외처리라고 하면 단순하게 '그냥 try-catch하면 되는 게 아닌가?'라고 쉽게 생각할 수도 있지만 사실 이야기를 하자면 상당한 설계 차원의 지식이 필요한 문제가 예외처리입니다.

초보 개발자 분들이라면 그렇게 깊게까지 파고들 필요는 없겠지만 최소한 스택트레이스는 정확히 읽을 줄 알아야만 빠른 디버깅이 가능한 만큼 이 부분에 대해서만은 시간을 투자해서 정확한 지식을 쌓아둘 필요가 있다고 생각합니다.

이번 면접에서 제가 제시한 스택 트레이스는 이렇습니다 :

```
java.lang.reflect.InvocationTargetException
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
     at java.lang.reflect.Method.invoke(Unknown Source)
     at com.mylibrary.ap.xul.builder.handler.MethodCallback.invokeCallback(MethodCallback.java:32)
     at com.mylibrary.ap.xul.builder.handler.ViewHandler.invokeActionResult(ViewHandler.java:581)
     at com.mylibrary.ap.xul.context.impl.ViewContextImpl$2.onActionResult(ViewContextImpl.java:283)
     at com.mylibrary.ap.xul.context.impl.ActionContextImpl.setResult(ActionContextImpl.java:90)
     at com.mylibrary.ap.xul.action.stock.DialogAction.handleDialogClose(DialogAction.java:156)
     at com.mylibrary.ap.xul.action.stock.DialogAction$1.windowClosed(DialogAction.java:142)
     at com.mylibrary.api.Window.fireWindowClosedEvent(Unknown Source)
     at com.mylibrary.api.Window.onClose(Unknown Source)
     at com.mylibrary.api.platform.WindowBase.BaseClose(Native Method)
     at com.mylibrary.api.platform.WindowBase.BaseClose(Unknown Source)
     at com.mylibrary.api.Window.close(Unknown Source)
     at com.mylibrary.ap.xul.ui.MessageDialog.handleButtonClick(MessageDialog.java:145)
     at com.mylibrary.ap.xul.ui.MessageDialog$1.handleClick(MessageDialog.java:134)
     at com.mylibrary.api.ClickableSupport.fireClickEvent(Unknown Source)
     at com.mylibrary.api.ClickableSupport.handleAction(Unknown Source)
     at com.mylibrary.api.Button.actionHook(Unknown Source)
     at com.mylibrary.api.Component.action(Unknown Source)
Caused by: java.lang.NullPointerException
     at com.mycompany.service.impl.PortalManagerImpl.deleteMenuItem(PortalManagerImpl.java:603)
     at com.mycompany.service.impl.PortalManagerImpl.deletePortal(PortalManagerImpl.java:358)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
     at java.lang.reflect.Method.invoke(Unknown Source)
     at org.springframework.aop.support.AopUtils.invokeJoinpointUsingReflection(AopUtils.java:307)
     at org.springframework.aop.framework.ReflectiveMethodInvocation.invokeJoinpoint(ReflectiveMethodInvocat
     at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java
     at org.springframework.transaction.interceptor.TransactionInterceptor.invoke(TransactionInterceptor.jav
     at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java
     at org.springframework.security.intercept.method.aopalliance.MethodSecurityInterceptor.invoke(MethodSec
     at org.springframework.aop.framework.ReflectiveMethodInvocation.proceed(ReflectiveMethodInvocation.java
     at org.springframework.aop.framework.JdkDynamicAopProxy.invoke(JdkDynamicAopProxy.java:204)
     at $Proxy54.deletePortal(Unknown Source)
     at sun.reflect.NativeMethodAccessorImpl.invoke0(Native Method)
     at sun.reflect.NativeMethodAccessorImpl.invoke(Unknown Source)
     at sun.reflect.DelegatingMethodAccessorImpl.invoke(Unknown Source)
     at java.lang.reflect.Method.invoke(Unknown Source)
     at com.mycompany.util.SpringSecurityContextInvocationHandler.invoke(SpringSecurityContextInvocationHand
     at $Proxy84.deletePortal(Unknown Source)
     at com.mycompany.ui.binding.PortalDataProvider.doDelete(PortalDataProvider.java:81)
     at com.mycompany.ui.binding.PortalDataProvider.doDelete(PortalDataProvider.java:12)
     at com.mycompany.ui.binding.AbstractEISDataProvider.delete(AbstractEISDataProvider.java:105)
     at com.mylibrary.ap.xul.binding.dataset.impl.DatasetImpl.doCommit(DatasetImpl.java:90)
     at com.mylibrary.ap.xul.binding.dataset.impl.AbstractDataset.commit(AbstractDataset.java:251)
     at com.mylibrary.ap.xul.binding.dataset.impl.AbstractDataset.deleteRow(AbstractDataset.java:201)
     at com.mylibrary.ap.xul.action.dataset.DeleteDataRowAction.execute(DeleteDataRowAction.java:22)
     at com.mylibrary.ap.xul.context.impl.ViewContextImpl.execute(ViewContextImpl.java:294)
     at com.mycompany.ui.portal.PortalInfoHandler.onPostConfirmDeleteAction(PortalInfoHandler.java:192)
     ... 21 more
```

상당히 길죠? 제가 본 많은 수의 초보 개발자분들은 이런 로그를 접하면 지레 겁을 먹고 정확하게 트레이스를 읽으려 들지 않습니다.

사실 예외를 읽는 법은 기초 중에서도 가장 중요한 내용이고, 처음 자바 언어를 배우는 모든 개발자가 반드시 숙지해야할 내용인데도 오히려 제대로 아는 초급 개발자를 만나기 힘든 것을 보면 도대체 자바 학원이라는 곳에선 무엇을 가르치는 건가 의아한 생각이 듭니다.

오류가 발생하면 무조건 오류 메시지부터 완벽하게 이해해야합니다. 왜 책에서 본대로 쳤는데 안되나, 왜 어제까지 됐는데 갑자기 안될까, 아니면 인터넷에서 한 번 찾아볼까 등의 생각은 일단 오류부터 제대로 읽어보고 해도 늦지 않습니다.

예외처리를 제대로 작성한 코드라면 거의 대부분의 문제는 스택트레이스 안에 답이 있기 때문입니다.

우선 위의 예의 경우 트레이스가 두 가지 예외 내용을 포함하고 있다는 것을 바로 파악할 수 있어야 하며, 문제의 진정한 원인은 윗쪽의 트레이스가 아니라 '*Caused By:*'로 표시되는 널포인터 예외 부분임을 인지할 수 있어야 합니다.

일단 문제에 대한 원인이 되는 트레이스를 찾았다면 그 다음은 트레이스를 읽어야 하는데, 이제까지 꽤 많은 수의 초급 개발자들의 면접을 경험했지만 의외로 트레이스 읽는 법을 정확하게 아는 경우가 드물었습니다.

면접시 위의 트레이스를 보여주고 어쩔 줄 몰라할 경우 보통 힌트를 주고 예를들어 널포인터 예외가 찍힌 부분부터 아래쪽으로 단 세줄만 집중해서 무슨 뜻인지 해석해보라고 질문합니다. 즉,

```
Caused by: java.lang.NullPointerException
     at com.mycompany.service.impl.PortalManagerImpl.deleteMenuItem(PortalManagerImpl.java:603)
     at com.mycompany.service.impl.PortalManagerImpl.deletePortal(PortalManagerImpl.java:358)
```

이 부분을 보여주고 읽는 방법을 알고 있는지 묻는 질문인데, 이를 정확하게 '*com.mycompany.service.impl.PortalManagerImpl*' 클래스의 '*deletePortal*' 메소드 358라인에서 같은 클래스의 '*deleteMenuItem*'메소드를 호출했는데 해당 메소드 603번 째 줄에서 널포인터 예외가 발생했다'라고 해석할 수 있는 지원자가 많지 않았습니다.

아예 대답을 못하거나 트레이스를 위에서 아래로 해석하려 하거나 심지어 이를 '두 번의 널포인터 예외가 발생했다'로 해석하는 경우까지 있더군요. 이런 로그를 보고 곧바로 해당 클래스의 603번 째 줄을 찾아갈 수 있는 개발자와 '내가 뭘 잘못 쳐서 안되는 거지?' 하는 식으로 막연한 고민으로 시간을 허비하는 개발자를 상상해보시면 스택트레이스를 정확히 아는 것이 얼마나 디버깅과 관련한 개발생산성에 큰 도움을 주는 지 짐작 가능하시리라 생각합니다.

그리고 '*PortalManagerImpl*' 클래스의 관련된 소스는 다음과 같았습니다:

```
if (item == null) {
    throw new NullArgumentException("item");
}

//중간 생략
List<PortalMenu> children = getMenuItems(item.getPortal().getId(), item.getId()); // 603번째 줄

for (PortalMenu child : children) {
    deleteMenuItem(child);
 }
```

신기하게도 아직까지 면접을 본 개발자 중에 이런 문제를 보고 곧바로 널포인터 예외의 원인을 짚어내는 지원자는 거의 없었습니다. 글을 읽으시는 분들은 모두 금방 답이 보이시나요?

많은 수의 지원자들이 '*children*'이나 '*item.getId()*' 등에 널값이 들어간 것 같다고 답했습니다. 이론적으로 해당 라인에서 널값이 들어갈 수 있는 모든 경우의 수는,

1. *children*
2. *item*
3. *item.getPortal()*
4. *item.getPortal().getId()*
5. *item.getId()*

이렇게 다섯 가지가 전부입니다.

이 중 적어도 두 가지, 즉 2번 혹은 3번으로 가능성을 바로 좁히지 못한다면 그것은 널포인터 예외의 의미를 정확하게 파악하지 못하고 있기 때문입니다.

널포인터 예외는 단순하게 변수에 널값이 들어가서 생기는 오류가 아닙니다. 널포인터 예외는 명확하게 객체의 널레퍼런스에 대해 메소드 호출이나 필드 참조 등의 작업을 했을 때 발생하는 문제라는 것을 이해한다면 이런 문제는 곧바로 원인을 좁힐 수 있어야 합니다.

즉, 1번의 경우처럼 단순히 변수에 널값을 할당하는 것만으로는 절대로 널포인터 예외가 날 수 없습니다. 그리고 만일 4 번 '*item.getPortal().getId()*'이나 5번 '*item.getId()*'이 널이라면 이는 널 레퍼런스에 대한 호출이 아니라 널값을 '*getMenuItems*'라는 메소드의 인자로 넘기는 것 뿐이기 때문에 역시 널포인터 예외의 원인이 될 수 없습니다.

물론 '*getMenuItem*' 메소드 안에서 해당 인자에 대한 널체크 없이 값을 사용하다가 예외가 날 수도 있겠지만 이 경우엔 절대로 트레이스 상에 굵은 글자로 표시된 603번 째에서 예외를 뿌리지 않습니다.

그렇다면 남은 가능성은 2번 '*item*'이 널이거나 3번 '*item.getPortal()*'이 널인 경우뿐인데, '*item*' 변수는 위에서 널체크를 하기 때문에 603번 째 줄에서 절대로 널값을 가질 수 없습니다. 그렇기 때문에 답은 3번이 되는 것입니다.

너무 문제가 어렵나요? 길게 설명했지만 개발을 하면 매우 흔하게 접할 수 있는 예외이며 따져보면 결코 어려운 내용이 아닙니다. 위와 같은 문제는 이미 스택트레이스를 능숙하게 읽을 수 있고 널포인터 예외를 이해하고 있는 개발자라면 몇 초 안에 곧바로 원인을 파악할 수 있는 문제입니다.

예외만 읽을 줄 알면 단 몇 초만에 깨달을 수 있는 문제를 다른 개발자는 인터넷 검색과 의미없는 무작위 수정, 게시판 질문글 작성등으로 몇 시간씩 허비한다면 두 개발자 사이의 생산성의 차이는 엄청나게 벌어질 것입니다.

디버깅 같은 기본적 부분에서 나쁜 습관을 빨리 고치지 못한다면 시간 낭비도 문제지만 그 만큼 개발자로서 발전도 늦게되고, 나중에는 인터넷이 없으면 아무 것도 못하는 그냥 복사 붙여넣기만 단순 코더가 한계점이 될 수도 있습니다.

그런 점을 감안해서 이 글을 보는 초보 개발자들은 반드시 제대로된 문제 해결 방법을 공부하셨으면 좋겠습니다.

댓글 35

fender 13k
2016-07-18 20:48:08

4

아울러, 일반적인 문제 해결에 대한 접근 방법으로는 아래 글도 참고하시면 도움이 될 것 같습니다:

- "개발은 암기과목이 아닙니다"

fender ϟ 13k
2016-01-31 11:05:25

33 21884

#311337 포럼 신입 조언 기초 접근방법 디버깅

개발은 암기과목이 아닙니다

65

57

가끔씩 질문 게시판에 들러 글을 읽다보면 오류 내용을 통째로 복사해서 붙여 넣고 "이런 오류가 생겼는데 어떻게 해야 하나요?" 와 같은 식의 질문이 너무 많은 것 같아 안타까운 마음이 듭니다.

개발은 절대 암기로 배울 수 있는 영역이 아닙니다. 어떤 메시지가 나오면 어떤 부분을 고치면 된다는 식의 사례를 많이 외운다고 디버깅 능력이 높아지지는 않습니다.

프로그래밍 과정에서 발생하는 오류 메시지는 복사해서 검색을 하라고 나오는 것이 아니라 읽고 원인에 대해 생각하라고 제공되는 것입니다. 디버깅이 이런 경우에 이렇게 고치고 저런 경우에 저렇게 고치는 공식을 외워서 적용하는 과정이라면 오류 사전 같은 게 벌써 나왔겠지요.

예를들어, 커넥션 풀을 사용하다 "*Connection Pool Exception: Cannot get a connection, pool error Timeout waiting for idle object*"와 같은 오류를 접했다면, 가장 먼저 할 일은 복사해서 구글에서 검색하거나 질문 답변 게시판에 "이런 오류 났는데 어떻게 해야되요?"라고 묻는 것이 아니라 오류 내용을 읽어보는 것입니다.

그래서 '커넥션 풀', '타임아웃', '아이들 객체' 같은 개념들이 어떤 의미를 가지는지 이해하려고 노력하고, 안되면 관련 문서를 찾아보는 과정을 거치는 것이 정상입니다.

'커넥션 풀'이 무슨 개념이고 왜 사용하는지를 이해했다면 아마도 '아이들 객체'가 대기 상태의 데이터베이스 연결을 뜻함은 어렵지 않게 유추할 수 있을 것입니다. 스택트레이스를 읽을 줄 안다면 해당 오류가 '*commons-dbcp*'라는 커넥션 풀 라이브러리에서 발생하는 것임도 알아볼 수 있을테니 필요하면 해당 라이브러리의 프로젝트 페이지에 찾아가 문서를 참조할 수도 있을 것입니다. (인터넷 검색은 보통 스택트레이스를 통째로 검색하는 게 아니라 이런 경우 'dbcp'라는 키워드로 프로젝트 페이지를 찾을 때 쓰는 것입니다.)

이렇게 기본 개념들을 다시 정리했으면 이젠 왜 그런 문제가 생겼는지 연역적 추리를 통해 원인을 찾아보면 됩니다.

타임아웃이 될 때까지 대기 상태의 연결을 얻지 못하고 포기했다는 건 어떤 경우에 발생할까 생각해보면 어렵지 않게 '모든 연결이 타임아웃 시간 동안 사용 중인 경우'가 원인임을 유추할 수 있습니다. 그럼 다음 단계로, 어떤 경우에 타임아웃 시간 동안 연결을 항상 사용하게 되는 지 고민해보면 대략 다음과 같은 논리적 '경우의 수'가 발생하는 것을 알 수 있습니다:

1. 연결을 한 번에 너무 오래 사용하는 경우
2. 커넥션 풀이 예약하는 연결 갯수가 지나치게 적은 경우.
3. 지나치게 짧은 시간 동안 대기하는 경우.

이런 경우 이해가 가지 않으면 머릿속으로 비슷한 경우를 형상화하거나 비유해서 이해해보는 것이 도움이 됩니다. 예컨대 커넥션 풀의 자원 관리 모델은 은행을 방문해서 대기표를 받고 업무를 보는 것과 비슷합니다.

은행 업무를 보기 위해 들렀다가 사람이 너무 많아서 점심시간 내내 기다리다 되돌아갔다면 이유가 무엇이었을지는 아마 은행 사진을 찍어서 게시판에 질문하지 않아도 누구나 쉽게 이해할 수 있을 것이라 생각합니다.

점심시간이라는 정해진 기간이 타임아웃 기간이고, 은행 창구가 얻으려는 리소스, 즉 데이터베이스 연결이고, 해당 지점 창구의 갯수가 커넥션 풀의 크기('max idle')라면 앞서 발생한 오류의 원인이 되는 경우의 수를 직관적으로 이해할 수 있을 것입니다:

1. 연결을 한 번에 너무 오래 사용하는 경우 -> 예) 진상 고객.
2. 커넥션 풀이 예약하는 연결 갯수가 지나치게 적은 경우 -> 창구나 담당 직원 부족.
3. 지나치게 짧은 시간 동안 대기하는 경우 -> 바빠서 빨리 사무실도 돌아가야하는 경우.

이렇게 경우의 수까지 추출했다면 그 다음으로는 실제 지금 프로젝트가 어느 경우에 해당하는지 검증만 하면 됩니다. 1번의 경우라면 정말 오래 걸리는 질의가 있는지, 혹은 질의는 짧게 끝나도 연결을 닫지 않아 계속 사용 중 상태로 남아있는 경우가 있는지 찾아보면 되겠지요.

사실 이번 오류의 예는 얼마전 실제로 이곳 질문 게시판에서 읽은 사례입니다. 질문자는 역시 오류 메시지만 복사해서 질문을 올렸고 이에 대한 답은 "'*removeAbandoned*' 속성을 설정하면 되더라" 였습니다.

DBCP 라이브러리의 '*removeAbandoned*' 속성은 연결을 일정 시간 닫지 않았을 때 이를 강제로 해제하는 속성입니다. 이는, 앞서 말한 경우의 수 중 1번의 경우, 또 그 중에서도 질의가 오래 걸리는 것이 아니라 개발자가 연결 해제를 잊었을 경우에만 적용될 수 있는 해법입니다. 그나마도, 정상적인 해결 방법은 해당 부분을 찾아서 연결 해제 코드를 넣는 것이기 때문에 임시 방편이나 미봉책에 불과하다고 할 수 있습니다.

이는 비유를 하자면, 내가 은행 지점장인데 점심시간에 사람이 붐벼서 고객 불만 사항이 많은 경우에 덮어놓고 "한 사람당 창구는 3분씩만 쓰게 하라"고 강제하는 것입니다. 실제로는 창구를 하나 더 열거나 창구 하나 잡고 한 시간씩 떼를 쓰는 진상 고객 한 명만 처리하면 될 문제를 덮어놓고 "창구가 붐비면 이용 시간을 제한한다"는 '해법'을 공식처럼 적용한 결과입니다.

그럼에도 불구하고 이런 식으로 문답이 오가는 것은 질문자나 답변자 모두 올바른 디버깅 방법을 이해하지 못하고, 그냥 어떤 오류가 발생하면 어떻게 대처한다는 단편적 지식을 많이 쌓으면 개발 능력이 높아질 것으로 기대하는 잘못된 인식을 가지고 있기 때문입니다.

그런 습관이 생긴 경우 흔히 보는 현상이 무작정 아무 것이나 바꾸어 보는 것입니다. 검색해서 해결이 안되면 지푸라기라도 잡는 심정으로 개발 환경도 새로 설치해보고, 어제는 된 것 같은데 오늘부터 안된다면 어제자 소스로 롤백해보기도 하는 식으로 시간 낭비를 하는 것은 흔하게 볼 수 있습니다.

역시 은행 지점의 예로 비유를 하자면 사람이 많아 불만이 접수되면 일단 인테리어도 바꾸어보고 그런 문제가 없는 지점과 직원들도 서로 교체해보고 하는 식의 쓸데없는 시간 낭비를 하는 것입니다.

혹시 해당 질문 답변 글을 쓰신 분들이 이 글을 보시고 기분이 상하셨다면 죄송합니다만, 저는 단지 해당 사례가 바로 기억에 떠올라서 예시로 들었을 뿐 이는 그 분들만의 문제도 아니고 그 분들의 잘못이라고만 할 수 있는 것도 아니라고 봅니다.

이 곳의 질문 답변 게시판만 봐도 단순히 오류 내용 복사해 붙여넣고 내용을 이해하려는 어떠한 노력 없이 "이런 오류가 생겼는 데 어떻게 하나요?"하고 물어보는 글은 너무도 흔하게 접할 수 있습니다. 그리고 "이 것 저 것 다해봐도 안되요", "어제까진 됐는데 갑자기 안되요", "이클립스도 다시 깔아봤어요" 같은 절박한 푸념도 쉽게 보는 내용입니다.

제 생각에 이는 신입 개발자의 자질보단 그런 신입 개발자를 키워내는 시스템의 문제가 더 크지 않나 싶습니다. 개발자를 양성하는 학원은 많지만 개념이나 접근 방법을 제대로 가르치기보단 단 시간에 스프링 MVC 예제 따라해보기 같은 식으로만 수업을 진행하니 정작 중요한 내용은 수박 겉핥기 식으로 넘어가게 된 것이 아닌가 생각합니다.

또한 실무에서도 회사들이 면접에서 "스프링은 써봤어요?" 같은 걸 물어보지 디버깅을 어떻게 하는지, 객체지향을 얼마나 이해하고 있는지 같은 근본적인 지식을 따지지 않기 때문에 개발자로 입문하는 분들이 무턱대고 준비없이 완전히 이해하지 못하는 코드를 따라 치는 연습만 하니까 원래 개발은 그런 식으로 하는 것이다라는 착각을 하게 되는 게 아닌가 싶습니다.

정리하면, 개발은 원리를 이해하고 이를 적용해서 문제를 해결하는 최적의 답을 찾는 과정이지 '스프링-마이바티스' 같은 정해진 답 하나를 반복 숙달해서 아무데나 가져다 쓰는 단순 반복 작업이 아닙니다.

또한 디버깅은 문제의 원인을 연역적 사고 과정을 통해 찾아내는 절차이지, 결코 "A의 오류 메시지에는 B로 대응"과 같은 공식을 찾아 적용하는 작업이 아닙니다.

혹시라도 그런 식의 잘못된 접근 방법으로 개발에 입문한 신입 개발자분들이 계시면 지금부터라도 올바른 방법이 무엇인지 고민해 보셨으면 좋겠습니다.

그래서 앞으로는 질문 답변 게시판에 "이런 오류가 났는데 도와주세요" 같은 글보다는 최소한 "이런 오류가 났는데 메시지에서 이런 개념이 이해가 가지 않습니다" 또는 "이런 오류는 이럴 때 발생하는 것 같은 데 다른 경우가 있을까요?" 같은 최소한의 사고 과정이 드러나는 질문을 보다 많이 볼 수 있게 되었으면 좋겠습니다.