スタックトレースの読み方

2016/3/24版　今泉 俊幸

目次

スタックトレースとは

- プログラムの実行過程のこと
  - 単にスタックトレースと言ったときは「例外スタックトレース」を指すことが多い
    - プログラムが例外を出して止まったときの実行過程
  - 不具合を修正する際、スタックトレースは原因箇所の特定の大きなヒントとなる
    - 逆にスタックトレースがないとかなり困難
- スタックトレースの「スタック」とはデータ構造のスタックの事
  - 実はプログラムのメソッド呼び出しはスタックで実現されている
    - メソッドを呼び出したらpush、実行が終わったらpop

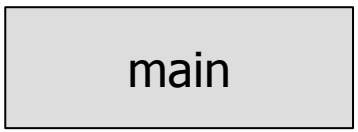
スタックとメソッド呼び出し

サンプルコード

プログラム実行位置

```
package jp.co.bbreak;

public class MyReader {
  public static void main(String[] args) {
    new MyReader().hoge();
  }

 public void hoge(){
    piyo();
    fuga();
  }

  private void piyo(){
  }

  private void fuga(){
    Reader reader = new BufferedReader(null);
  }
```

メソッド呼び出しの過程

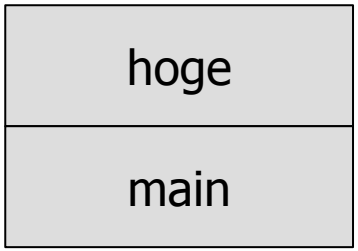


スタックとメソッド呼び出し

サンプルコード

```
package jp.co.bbreak;

public class MyReader {
  public static void main(String[] args) {
    new MyReader().hoge();
  }

 public void hoge(){
    piyo();
    fuga();
  }

  private void piyo(){
  }

  private void fuga(){
    Reader reader = new BufferedReader(null);
  }
```

メソッド呼び出しの過程



スタックとメソッド呼び出し

サンプルコード

```
package jp.co.bbreak;

public class MyReader {
  public static void main(String[] args) {
    new MyReader().hoge();
  }

 public void hoge(){
    piyo();
    fuga();
  }

  private void piyo(){
  }

  private void fuga(){
    Reader reader = new BufferedReader(null);
  }
```

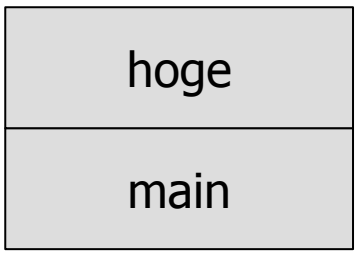
メソッド呼び出しの過程

スタックとメソッド呼び出し

サンプルコード

```
package jp.co.bbreak;

public class MyReader {
  public static void main(String[] args) {
    new MyReader().hoge();
  }

 public void hoge(){
    piyo();
    fuga();
  }

  private void piyo(){
  }

  private void fuga(){
    Reader reader = new BufferedReader(null);
  }
```

メソッド呼び出しの過程



スタックとメソッド呼び出し

サンプルコード

```
package jp.co.bbreak;

public class MyReader {
  public static void main(String[] args) {
    new MyReader().hoge();
  }

 public void hoge(){
    piyo();
    fuga();
  }

  private void piyo(){
  }

  private void fuga(){
    Reader reader = new BufferedReader(null);
  }
```

メソッド呼び出しの過程

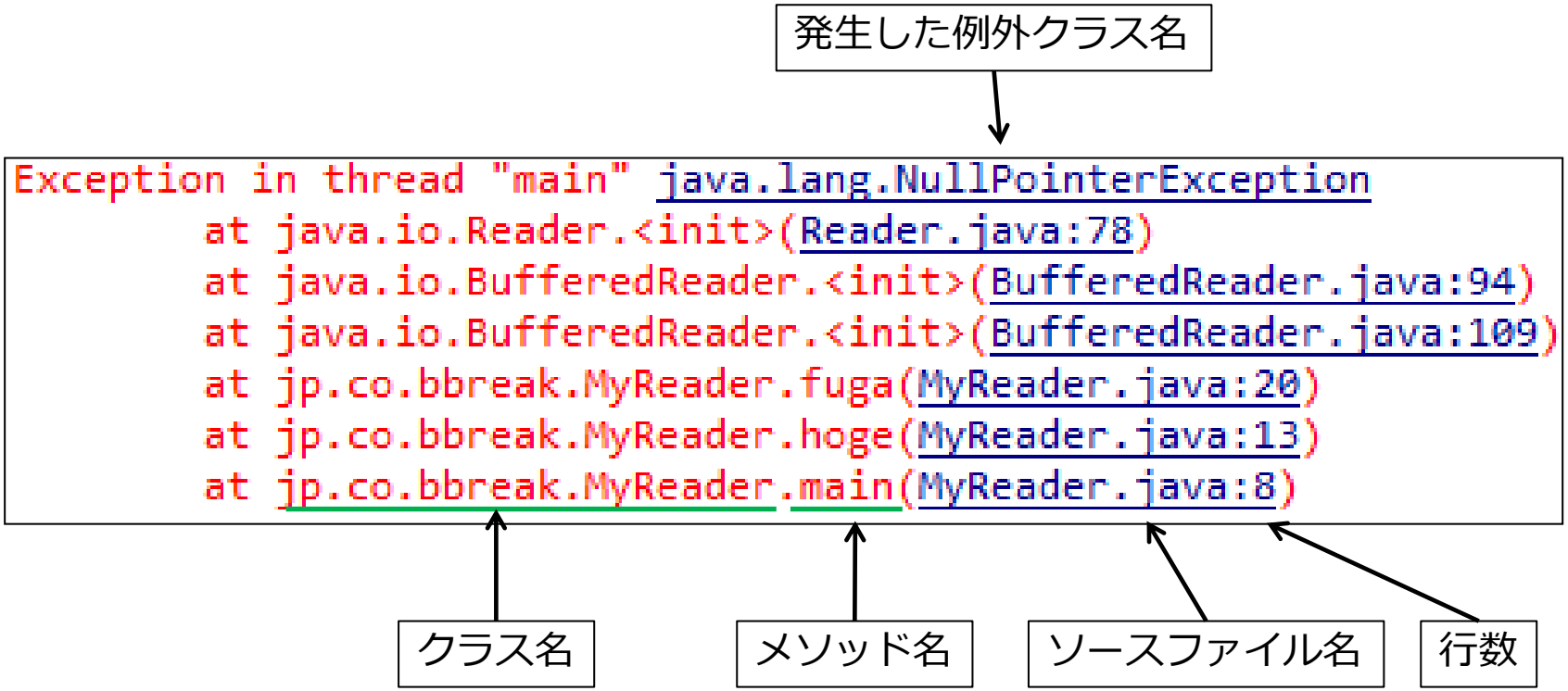
スタックトレースの例

```
Exception in thread "main" java.lang.NullPointerException
        at java.io.Reader.<init>(Reader.java:78)
        at java.io.BufferedReader.<init>(BufferedReader.java:94)
        at java.io.BufferedReader.<init>(BufferedReader.java:109)
        at jp.co.bbreak.MyReader.fuga(MyReader.java:20)
        at jp.co.bbreak.MyReader.hoge(MyReader.java:13)
        at jp.co.bbreak.MyReader.main(MyReader.java:8)
```

以下の順でメソッド呼び出しが行われた事が分かる

main
→hoge
→fuga
→BufferedReader
→Reader

スタックトレースの例



スタックトレースを追う 1/3

- まず原因を確認する

```
Exception in thread "main" java.lang.NullPointerException
        at java.io.Reader.<init>(Reader.java:78)
        at java.io.BufferedReader.<init>(BufferedReader.java:94)
        at java.io.BufferedReader.<init>(BufferedReader.java:109)
        at jp.co.bbreak.MyReader.fuga(MyReader.java:20)
        at jp.co.bbreak.MyReader.hoge(MyReader.java:13)
        at jp.co.bbreak.MyReader.main(MyReader.java:8)
```

- NullPointerExceptionが発生したことが原因
  - nullのオブジェクトに対してメソッドが呼び出された
  - 例外の意味が分からなかったら検索サイト等で調べる

スタックトレースを追う 2/3

- エラーの原因箇所を追う

```
Exception in thread "main" java.lang.NullPointerException
        at java.io.Reader.<init>(Reader.java:78)
        at java.io.BufferedReader.<init>(BufferedReader.java:94)
        at java.io.BufferedReader.<init>(BufferedReader.java:109)
        at jp.co.bbreak.MyReader.fuga(MyReader.java:20)
        at jp.co.bbreak.MyReader.hoge(MyReader.java:13)
        at jp.co.bbreak.MyReader.main(MyReader.java:8)
```

- スタックの一番上が例外が発生した箇所
- javaパッケージであり、自分が実装した箇所ではない。
  自分が実装したクラスが出てくるまで下にたどる
  - 今回の例だとMyReader.fugaメソッド

スタックトレースを追う 3/3

- コードを見てみる

```
package jp.co.bbreak;

public class MyReader {
  public static void main(String[] args) {
    new MyReader().hoge();
  }

 public void hoge(){
    piyo();
    fuga();
  }

  private void fuga(){
    Reader reader = new BufferedReader(null);
  }
```

nullを渡していることが原因！

発展：ネストした例外

- 例外はネストすることがある

```
private void fuga(String number){
    try{
      Integer.parseInt(number);
    }catch (NumberFormatException e) {
      throw new IllegalArgumentException(e);
    }
}
```

- NumberFormatExceptionをIllegalArgumentExceptionでラップしている
- 発生した例外を別の例外として扱いたい場合にこのような実装をする
  - 検査例外を非検査例外にする場合などありがち

ネストした例外のスタックトレース

```
Exception in thread "main" java.lang.IllegalArgumentException: java.lang.NumberFormatException: For input string: "hoge"
        at jp.co.bbreak.MyReader.fuga(MyReader.java:25)
        at jp.co.bbreak.MyReader.hoge(MyReader.java:14)
        at jp.co.bbreak.MyReader.main(MyReader.java:9)
Caused by: java.lang.NumberFormatException: For input string: "hoge"
        at java.lang.NumberFormatException.forInputString(NumberFormatException.java:65)
        at java.lang.Integer.parseInt(Integer.java:492)
        at java.lang.Integer.parseInt(Integer.java:527)
        at jp.co.bbreak.MyReader.fuga(MyReader.java:23)
        ... 2 more
```

- ネストした例外が先に表示される
- 本当の原因となる例外はCause by以下に表示されるので、こちらを参照する

まとめ

- 例外スタックトレースはプログラムの不具合の原因特定に非常に有益
- 例外スタックトレースには、発生した例外と、発生するまでのメソッド呼び出しの過程が含まれる
- スタックトレースを追う時は、上から順に見ていき、最初に自分の書いたコードが出てくるまでたどる