

## | 今天学习目标

- \* 能够掌握Runtime常见的方法

- \* 通过查看源码初步理解单例模式

- \* 手写单例

- \* 1 构造器私有

- \* 2 静态方法返回引用

- \* 3 静态变量构建这个对象

- \* getRuntime , freeMemory , totalMemory , maxMemory , availableProcessors

- \* exec ( mspaint,notepad,write )

- \* 能够理解异常的概述

- \* Throwable , Error ( OutOfMemoryError ) , Exception ( RuntimeException , IOException )

- \* RuntimeException

- \* InputMismatchException

- \* NullPointerException

- \* ArrayIndexOutOfBoundsException

- \* ArithmeticException

- \* ....

- \* 能够掌握异常的5个关键字try , catch , finally , throw,throws

- \* try{ }catch(小){}catch(大){}finally{}

- \* throw在方法里抛异常

- \* throws 在方法签名

- \* try return

- \* 4个面试题

```
try{  
    int i=1;  
    return i;
```

```

        // throw new RuntimeException();

    }catch(Exception e){

    }finally{

        // i=10;

        sysou("");

    }

```

\* 能够掌握Java的异常体系

\* 能够掌握自定义异常

\* 能够掌握日志框架log4j和log4j2

---

\* 回顾

\* 常见API

\* Object

\* native ( JNI ) ,

equals,toString,clone,finalize,hashCode,getClass,notify,notifyAll,wait

\* DecimalFormat

double d=8888.66666; 0.1-->10% "##.##%"

\* BigInteger,BigDecimal

\* 日期常见的类

\* Date , Calendar,SimpleDateFormat

\* LocalDate,LocalTime,LocalDateTime,DateTimeFormatter

\* String,Scanner,Math,Random,Arrays ,Byte,Short,Integer,Long,Character,Double...

\* 能够掌握Runtime常见的方法

\* 通过查看源码初步理解单例模式

```

1 // 1 构造器私有
2 // 2 提供一个静态的方法:getInstance,getRuntime
3 // 返回一个单例对象
4 // 3 可以提前先准备好
5 public class Singleton {
6     // 3 准备好这个对象（只加载一次）
7     private static Singleton singleton=new Singleton();
8     // 1 构造器私有
9     private Singleton() {
10
11     }
12     // 2 写一个静态方法
13     public static Singleton getInstance() {
14         return singleton;
15     }
16 }
17
18 Singleton instance1 = Singleton.getInstance();
19 Singleton instance2 = Singleton.getInstance();
20 System.out.println(instance1==instance2);
21 结果:
22 true

```

\* getRuntime , freeMemory , totalMemory , maxMemory , availableProcessors

\* exec ( mspaint,notepad,write )

```

1 public static void main(String[] args) {
2     // Runtime.getRuntime();
3     // int[] max=new int[100000000];
4     // Runtime.getRuntime().gc();
5     //获取可用内存
6     long value = Runtime.getRuntime().freeMemory();
7     System.out.println("可用内存为:"+value/1024/1024+"MB");
8     //获取jvm内存总数量，该值会不断的变化
9     long totalMemory = Runtime.getRuntime().totalMemory();
10    System.out.println("全部内存为:"+totalMemory/1024/1024+"MB");
11    //获取jvm 可以最大使用的内存数量，如果没有被限制 返回 Long.MAX_VALUE;

```

```

12     long maxMemory = Runtime.getRuntime().maxMemory();
13     System.out.println("可用最大内存为:"+maxMemory/1024/1024+"MB");
14     //获得jvm运行可用核数
15     int v = Runtime.getRuntime().availableProcessors();
16     System.out.println(v);
17     // 执行系统的命令
18     try {
19         Runtime runtime = Runtime.getRuntime();
20         //mspaint,notepad,write
21         runtime.exec("notepad");
22     } catch (Exception e) {
23         e.printStackTrace();
24     }
25 }
26 结果:
27 可用内存为:295MB
28 全部内存为:299MB
29 可用最大内存为:4425MB
30 8

```

\* 能够理解异常的概述

\* 案例

```

public static void main(String[] args) {
    Scanner input=new Scanner(System.in);
    try {
        System.out.println("输入年龄: ");
        int age=input.nextInt();//java.util.InputMismatchException:类型不匹配异常
        System.out.println(age);
    } catch (InputMismatchException e) { //类型不匹配异常
        System.err.println("类型不匹配");//error
        System.out.println("输入年龄: ");
        input=new Scanner(System.in);
        int age=input.nextInt();
        System.out.println(age);
    }
}

```

```

1 public static void main(String[] args) {
2     Scanner input=new Scanner(System.in);

```

```

3      try {
4          System.out.println("输入年龄: ");
5          int age=input.nextInt();//java.util.InputMismatchException:类型不匹配异常
6          System.out.println(age);
7      }catch (InputMismatchException e) { //类型不匹配异常
8          System.err.println("类型不匹配");//error
9          System.out.println("输入年龄: ");
10         input=new Scanner(System.in);
11         int age=input.nextInt();
12         System.out.println(age);
13     }
14 }

```

```

1 public class Test2 {
2     public static void main(String[] args) {
3         int a=12;
4         int b=0;
5         int result=a/b;
6         System.out.println(result);
7     }
8 }
9
10
11

```

Problems @ Javadoc Declaration Search Console

<terminated> Test2 (10) [Java Application] C:\Program Files\Java\jre1.8.0\_181\bin\javaw.exe (2018年8月23日 上午9:25:08)

Exception in thread "main" java.lang.ArithmeticException: / by zero  
at com.hx.exception.Test2.main(Test2.java:7)

```

1 public static void main(String[] args) {
2     int a=12;
3     int b=0;
4     int result=a/b;
5     System.out.println(result);
6 }

```

```

4 public static void main(String[] args) {
5     try {
6         int a=12;
7         int b=0;
8         int result=a/b;
9         System.out.println(result);
10    } catch (ArithmeticException e) {
11        e.printStackTrace();
12        System.out.println("被除数不能为零");
13    }
14 }
15 }

```

Problems @ Javadoc Declaration Search Console

<terminated> Test2 (10) [Java Application] C:\Program Files\Java\jre1.8.0\_181\bin\javaw.exe (2018年8月23日 上午9:29:43)

[java.lang.ArithmeticException](#): / by zero

被除数不能为零

at com.hx.exception.Test2.main([Test2.java:8](#))

```

1 public static void main(String[] args) {
2     try {
3         int a=12;
4         int b=0;
5         int result=a/b;
6         System.out.println(result);
7     } catch (ArithmeticException e) {
8         e.printStackTrace();
9         System.out.println("被除数不能为零");
10    }
11 }

```

```

public static void main(String[] args) {
    Scanner input=new Scanner(System.in);
    try {
        System.out.println("a:");
        int a=input.nextInt();
        System.out.println("b:");
        int b=input.nextInt();
        int result=a/b;
        System.out.println(result);
    }catch (ArithmeticException e) {
        e.printStackTrace();
        System.out.println("被除数不能为零");
    }finally {
        System.out.println("程序结束...");
    }
}

```

无论异常是否被捕获到，都会执行

```

1 public static void main(String[] args) {
2     Scanner input=new Scanner(System.in);
3     try {
4         System.out.println("a:");
5         int a=input.nextInt();
6         System.out.println("b:");
7         int b=input.nextInt();
8         int result=a/b;
9         System.out.println(result);
10    }catch (ArithmeticException e) {
11        e.printStackTrace();
12        System.out.println("被除数不能为零");
13    }finally {
14        System.out.println("程序结束...");
15    }
16
17 }

```

\* 能够掌握异常的5个关键字try , catch , finally , throw,throws

\* Java异常是Java提供的一种识别及响应错误的一致性机制。

\* try -- 用于监听。将要被监听的代码(可能抛出异常的代码)放在try语句块之内，当try语句块内发生异常时，异常就被抛出。

\* catch -- 用于捕获异常。catch用来捕获try语句块中发生的异常。

\* finally -- finally语句块总是会被执行。它主要用于回收在try块里打开的物力资源(如数据库连接、网络连接和磁盘文件)。只要有finally块，执行完成之后，才会回来执行try或者catch块中的return或者throw语句，如果finally中使用了return或者throw等终止方法的语句，则就不会跳回执行，直接停止。

\* throw -- 用于抛出异常。

• throws -- 用在方法签名中，用于声明该方法可能抛出的异常

```
4
5 public int devide(int a,int b) {
6     try {
7         int result=a/b;
8         return result;
9     } catch (Exception e) {
10        e.printStackTrace();
11        return 2;
12    }finally {
13        System.out.println("devide....");
14        return 3;
15    }
16 }
17
18 public static void main(String[] args) {
19     Test3 test3=new Test3();
20     int result=test3.devide(2, 0);
21     System.out.println("result:"+result);
22     System.out.println("main....");
23 }
```

只有finally块，执行完成之后，才会回来执行try或者catch块中的return或者throw语句，如果finally中使用了return或者throw等终止方法的语句，则就不会跳回执行，直接停止。

```
1 public int devide(int a,int b) {
2     try {
3         int result=a/b;
4         return result;
5     } catch (Exception e) {
6         e.printStackTrace();
7         return 2;
8     }finally {
9         System.out.println("devide....");
10    }
```



```

11     }
12
13     public static void main(String[] args) {
14         Test3 test3=new Test3();
15         int result=test3.devide(2, 1);
16         System.out.println("result:"+result);
17         System.out.println("main....");
18     }

```

```

public int devide(int a,int b) {
    int result=0;
    try {
        result=a/b;
    } catch (Exception e) {
        e.printStackTrace();
        // 处理...
        throw new ArithmeticException("被除数不能为0");
    }finally {
        System.out.println("devide....");
    }
    return result;
}

```

抛异常

```

1 public int devide(int a,int b) {
2     int result=0;
3     try {
4         result=a/b;
5     } catch (Exception e) {
6         e.printStackTrace();
7         // 处理...
8         throw new ArithmeticException("被除数不能为0");
9     }finally {
10        System.out.println("devide....");
11    }
12    return result;
13 }
14
15 public static void main(String[] args) {
16     Test4 test3=new Test4();

```

```

17     try {
18         int result=test3.devide(2, 0);
19         System.out.println("result:"+result);
20     }catch (ArithmeticException e) {
21         System.out.println("被除数不能为0");
22     }
23     System.out.println("main....");
24 }

```

常见面试题：

```

1 * 面试题一
2 public static void main(String[] args){
3     int result = test1();
4     System.out.println(result);
5 }
6
7 public static int test1(){
8     int i = 1;
9     try{
10         i++;
11         System.out.println("try block, i = "+i);
12     }catch(Exception e){
13         i--;
14         System.out.println("catch block i = "+i);
15     }finally{
16         i = 10;
17         System.out.println("finally block i = "+i);
18     }
19     return i;
20 }
21 * 结果
22 try block, i = 2
23 finally block i =10
24 10
25
26 * 面试题二
27 public static void main(String[] args){
28     int result = test2();

```

```

29     System.out.println(result);
30 }
31 public static int test2(){
32     int i = 1;
33     try{
34         i++;
35         throw new Exception();
36     }catch(Exception e){
37         i--;
38         System.out.println("catch block i = "+i);
39     }finally{
40         i = 10;
41         System.out.println("finally block i = "+i);
42     }
43     return i;
44 }

```

45  
46 答案:

```

47     catch block i =1
48     finally block i =10
49     10

```

50  
51 面试题三:

```

52 public static void main(String[] args){
53     int result = test3();
54     System.out.println(result);
55 }
56
57 public static int test3(){
58     //try 语句块中有 return 语句时的整体执行顺序
59     int i = 1;
60     try{
61         i++;
62         System.out.println("try block, i = "+i);
63         return i;
64     }catch(Exception e){
65         i ++;
66         System.out.println("catch block i = "+i);
67         return i;
68     }finally{

```

```
69     i = 10;
70     System.out.println("finally block i = "+i);
71 }
72 }
```

73

74

75 答案:

```
76     try block, i =2
77     finally block i =10
78     2
```

79 \* 备注:

80 你会发现在 `return` 语句返回之前，虚拟机会将待返回的值压入操作数栈，  
81 等待返回，即使 `finally` 语句块对 `i` 进行了修改，  
82 但是待返回的值已经确实的存在于操作数栈中了，所以不会影响程序返回结果。

83

84 面试题四

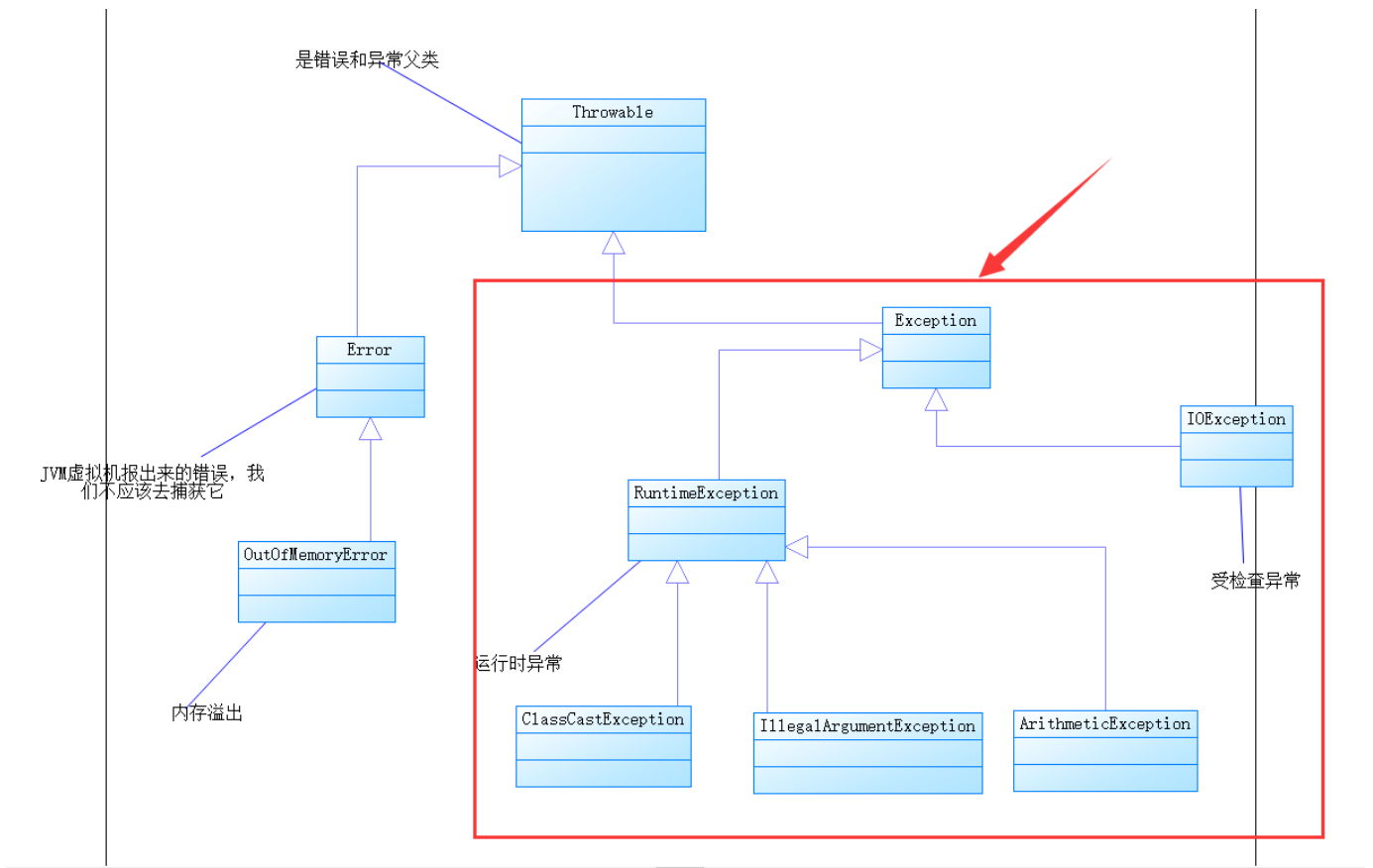
```
85 public static void main(String[] args){
86     int result = test4();
87     System.out.println(result);
88 }
89 public static int test4(){
90     //finally 语句块中有 return 语句
91     int i = 1;
92     try{
93         i++;
94         System.out.println("try block, i = "+i);
95         return i;
96     }catch(Exception e){
97         i++;
98         System.out.println("catch block i = "+i);
99         return i;
100     }finally{
101         i++;
102         System.out.println("finally block i = "+i);
103         return i;
104     }
105 }
```

106 结果:

```
107 try block, i =2
108 finally block i =3
```

109 3  
110 \* 参考链接  
111 \* <https://www.jianshu.com/p/49d2c3975c56>

\* 能够掌握Java的异常体系



\* Throwable :

The `Throwable` class is the superclass of all errors and exceptions in the Java language

- Object - java.lang
  - Throwable - java.lang
    - Error - java.lang
    - Exception - java.lang

\* Error

An `Error` is a subclass of `Throwable`

- \* that indicates serious problems that a reasonable application

- \* should not try to catch





















\* Exception

The class `Exception` and its subclasses are a form of

- \* `Throwable` that indicates conditions that a reasonable

- \* application might want to catch

#### Type hierarchy of 'java.lang.Throwable':

- >  `PrinterException` - java.awt.print
- >  `PrintException` - javax.print
  -  `PrivilegedActionException` - java.security
  -  `PropertySetterException` - org.apache.log4j.config
  -  `PropertySetterException` - org.apache.log4j.config
  -  `PropertyVetoException` - java.beans
  -  `REException` - sun.misc
  -  `ReflectiveCopyException` - com.sun.corba.se.spi.copyobject
- >  `ReflectiveOperationException` - java.lang
  -  `RefreshFailedException` - javax.security.auth
  -  `RemarshalException` - org.omg.CORBA.portable
  -  `RewriteException` - jdk.nashorn.internal.runtime
  -  `RuntimeException` - java.lang
- >  `SAXException` - org.xml.sax
- >  `SAXException` - jdk.internal.org.xml.sax
  -  `ScriptException` - javax.script
  -  `ServerNotActiveException` - java.rmi.server
  -  `ServerSideException` - com.sun.xml.internal.ws.developer
  -  `SnmpBadSecurityLevelException` - com.sun.jmx.snmp
  -  `SnmpSecurityException` - com.sun.jmx.snmp

\* RuntimeException

\* 能够掌握自定义异常

\* 能够掌握日志框架log4j和log4j2