JAVA常见异常

Java.io.NullPointerException

- null 空的，不存在的
- NullPointer 空指针

空指针异常，该异常出现在我们操作某个对象的属性或方法时，如果该对象是null时引发。

```
String str = null;
str.length();//空指针异常
```

上述代码中引用类型变量str的值为null，此时不能通过它调用字符串的方法或引用属性，否则就会引发空指针异常。

解决办法:

找到为什么赋值为null，确保该对象的值不能为null再操作属性或方法即可。

java.lang.NumberFormatException: For input string: "xxxxx"

- Number 数字
- Format 格式

数字格式异常，该异常通常出现在我们使用包装类将一个字符串解析为对应的基本类型时引发。

```
String line = "123.123";//小数不能转换为整数!
int d = Integer.parseInt(line);//抛出异常NumberFormatException
System.out.println(d);
```

上述代码中由于line的字符串内容是"123.123".而这个数字是不能通过包装类Integer解析为一个整数因此出现该异常。注:非数字的字符出在解析时也会出现该异常。

解决办法:

确保解析的字符串正确表达了基本类型可以保存的值

```
String line = "123";
int d = Integer.parseInt(line);
System.out.println(d);//123
```

java.lang.StringIndexOutOfBoundsException

- index 索引，下标
- Bounds 边界
- OutOfBounds 超出了边界

字符串下标越界异常。该异常通常出现在String对应的方法中，当我们指定的下标小于0或者大于等于字符串的长度时会抛出该异常。

```
String str = "thinking in java";
char c = str.charAt(20);//出现异常
System.out.println(c);
```

解决办法:

指定下标时的范围应当在>=0并且<=字符串的长度。

java.io.InvalidClassException

- Invalid 无效的
- Class 类

无效的类异常，该异常出现在使用java.io.ObjectInputStream在进行对象反序列化时在readObject()方法中抛出。这通常是因为反序列化的对象版本号与该对象所属类现有的版本号不一致导致的。

可以通过在类上使用常量:

```
static final long serialVersionUID = 1L;
```

来固定版本号，这样序列化的对象就可以进行反序列化了。

JAVA建议我们实现Serializable接口的类主动定义序列化版本号,若不定义编译器会在编译时根据当前类结构生成版本号,但弊端是只要这个类内容发生了改变,那么再次编译时版本号就会改变,直接的后果就是之前序列化的对象都无法再进行反序列化.

如果自行定义版本号,那么可以在改变类内容的同时不改变版本号,这样一来,反序列化以前的对象时对象输入流会采取兼容模式,即:当前类的属性在反序列化的对象中还存在的则直接还原,不存在的就是用该属性的默认值

出现该异常的解决办法:

1. 首先使用上述常量固定版本号
2. 重新序列化对象(将对象通过ObjectOutputStream重新序列化并写出)
3. 再进行反序列化即可

需要注意，之前没有定义序列化版本号时序列化后的对象都无法再反序列化回来，所以若写入了文件，可将之前的那些文件都删除，避免读取即可。

java.io.NotSerializableException

- NotSerializable 不能序列化

不能序列化异常，该异常通常出现在我们使用java.io.ObjectOutputStream进行对象序列化(调用writeObject)时。原因时序列化的对象所属的类没有实现java.io.Serializable接口导致

出现该异常的解决办法:

将序列化的类实现该接口即可

java.io.UnsupportedEncodingException

- Unsupported 不支持的
- Encoding字符集

不支持的字符集异常，该异常通常出现在使用字符串形式指定字符集名字时，犹豫字符集名字拼写错误导致。例如

```
PrintWriter pw = new PrintWriter("pw.txt", "UFT-8");
```

上述代码中，字符集拼写成"UFT-8"就是拼写错误。

常见的字符集名字:

- GBK:我国的国标编码，其中英文1个字节，中文2字节
- UTF-8:unicode的传输编码，也称为万国码。其中英文1字节，中文3字节。
- ISO8859-1:欧中的字符集，不支持中文。

java.io.FileNotFoundException

- File 文件
- NotFound 没有找到

文件没有找到异常，该异常通常出现在我们使用文件输入流读取指定路径对应的文件时出现

```
FileInputStream fis = new FileInputStream("f1os.dat");
```

上述代码如果指定的文件f1os.dat文件不在当前目录下，就会引发该异常:

java.io.FileNotFoundException: f1os.dat (系统找不到指定的文件。)

注:

抽象路径"f1os.dat"等同于"./f1os.dat"。因此该路径表示当前目录下应当有一个名为f1os.dat的文件。

还经常出现在文件输出流写出文件时，指定的路径无法将该文件创建出来时出现

```
FileOutputStream fos = new FileOutputStream("./a/fos.dat");
```

上述代码中，如果当前目录下没有a目录，那么就无法在该目录下自动创建文件fos.dat，此时也会引发这个异常。

其他API上出现该异常通常也是上述类似的原因导致的。

解决办法:

在读取文件时，确保指定的路径正确，且文件名拼写正确。

在写出文件时，确保指定的文件所在的目录存在。

java.net.ConnectException: Connection refused: connect

- connection 连接
- refused 拒绝

连接异常,连接被拒绝了.这通常是客户端在使用Socket与远端计算机建立连接时由于指定的地址或端口无效导致无法连接服务端引起的.

```
System.out.println("正在连接服务端...");
Socket socket = new Socket("localhost",8088);//这里可能引发异常
System.out.println("与服务端建立连接!");
```

解决办法:

- 检查客户端实例化Socket时指定的地址和端口是否正常
- 客户端连接前,服务端是否已经启动了

java.net.BindException: Address already in use

- bind 绑定
- address 地址
- already 已经
- Address already in use 地址已经被使用了

绑定异常,该异常通常是在创建ServerSocket时指定的服务端口已经被系统其他程序占用导致的.

```
System.out.println("正在启动服务端...");
ServerSocket serverSocket = new ServerSocket(8088);//这里可能引发异常
System.out.println("服务端启动完毕");
```

解决办法:

- 有可能是重复启动了服务端导致的,先将之前启动的服务端关闭
- 找到该端口被占用的程序,将其进程结束
- 重新指定一个新的服务端口在重新启动服务端

java.net.SocketException: Connection reset

- socket 套接字
- net 网络
- reset 重置

套接字异常,链接重置。这个异常通常出现在Socket进行的TCP链接时，由于远端计算机异常断开(在没有调用socket.close()的之前直接结束了程序)导致的。

解决办法:

- 无论是客户端还是服务端当希望与另一端断开连接时，应当调用socket.close()方法，此时会进行TCP的挥手断开动作。
- 这个异常是无法完全避免的，因为无法保证程序在没有调用socket.close()前不被强制杀死。

java.lang.InterruptedException

- interrupt 中断

中断异常.这个异常通常在一个线程调用了会产生阻塞的方法处于阻塞的过程中,此时该线程的interrupt()方法被调用.那么阻塞方法会立即抛出中断异常并停止线程的阻塞使其继续运行.

例如:

```
try {
    Thread.sleep(1000);
} catch (InterruptedException e) {
}
```

如果线程t1调用Thread.sleep(1000)处于阻塞的过程中,其他线程调用了t1线程的inerrupt()方法,那么t1调用的sleep()方法就会立即抛出中断异常InterruptedException并停止阻塞.

java.util.NoSuchElementException

- such　这个
- Element 元素

没有这个元素的异常.该异常通常发生在使用迭代器Iterator遍历集合元素时由于没有先通过hasNext()方法判断存在下一个元素而贸然通过next()获取下一个元素时产生(当集合所有元素都经过迭代器遍历一遍后还使用next获取).

```
while(it.hasNext()){
            String str = (String)it.next();
            //这里就可能产生NoSuchException异常
            System.out.println(it.next());
        }
```

上述代码中循环遍历时,每次调用hasNext()确定存在下一个元素时,循环里面连续调用过两次next()方法,这意味着第二次调用next()方法时并没有判断是否还存在.所以在最后会出现异常.

解决办法:

保证每次调用next()方法前都确定hasNext()为true才进行即可.

java.util.ConcurrentModificationException

Concurrent 并发

Modification 修改

并发修改异常.这个异常也经常出现在使用迭代器遍历集合时产生.

当我们使用一个迭代器遍历集合的过程中,通过集合的方法增删元素时,迭代器会抛出该异常.

```
while(it.hasNext()){
    //出现ConcurrentModificationException
    String str = (String)it.next();
    if("#".equals(str)){
        c.remove(str);//遍历过程中不要通过集合方法增或删元素
    }
    System.out.println(str);
}
```

解决办法:

使用迭代器提供的remove()方法可以删除通过next()获取的元素.

```
while(it.hasNext()){
            String str = (String)it.next();
            if("#".equals(str)){
//                c.remove(str);
               it.remove();
            }
            System.out.println(str);
        }
```

java.lang.UnsupportedOperationException

support 支持

unsupported 不支持的

operation 操作

不支持的操作异常.该异常出现在很多的API中.

例如:常出现在我们对数组转换的集合进行增删元素操作时抛出.

```
String[] array = {"one","two","three","four","five"};
System.out.println("array:"+ Arrays.toString(array));
List<String> list = Arrays.asList(array);//将数组转换为一个List集合
System.out.println("list:"+list);

list.set(0,"six");
System.out.println("list:"+list);
//对该集合的操作就是对原数组的操作
System.out.println("array:"+ Arrays.toString(array));

//由于数组是定长的,因此任何会改变数组长度的操作都是不支持的!
list.add("seven");//UnsupportedOperationException
```