1、抓抛模型的过程

过程一："抛"：程序在正常执行的过程中，一旦出现异常，就会在异常代码处生成一个对应异常类的对象。并将此对象抛出。一旦抛出对象以后，其后的代码就不再执行。

关于异常对象的产生：

① 系统自动生成的异常对象

② 手动的生成一个异常对象，并抛出（throw）

过程二："抓"：可以理解为异常的处理方式：① try-catch-finally ② throws

2、try-catch-finally的使用

1. 格式

```
try{
//可能出现异常的代码

}catch(异常类型1 变量名1){
//处理异常的方式1
}catch(异常类型2 变量名2){
//处理异常的方式2
}catch(异常类型3 变量名3){·
//处理异常的方式3
 }
 ....
 finally{
 //一定会执行的代码
 }
```

2. 说明

1. finally是可选的。

2. 使用try将可能出现异常代码包装起来，在执行过程中，一旦出现异常，就会生成一个对应异常类的对象，根据此对象的类型，去catch中进行匹配

3. 一旦try中的异常对象匹配到某一个catch时，就进入catch中进行异常的处理。一旦处理完成，就跳出当前的try-catch结构（在没有写finally的情况）。继续执行其后的代码

4. catch中的异常类型如果没有子父类关系，则谁声明在上，谁声明在下无所谓。catch中的异常类型如果满足子父类关系，则要求子类一定声明在父类的上面。否则，报错

5. 常用的异常对象处理的方式： ① String getMessage() ② printStackTrace()

6. 在try结构中声明的变量，再出了try结构以后，就不能再被调用

7. try-catch-finally结构可以嵌套

体会1：使用try-catch-finally处理编译时异常，是得程序在编译时就不再报错，但是运行时仍可能报错。相当于我们使用try-catch-finally将一个编译时可能出现的异常，延迟到运行时出现。

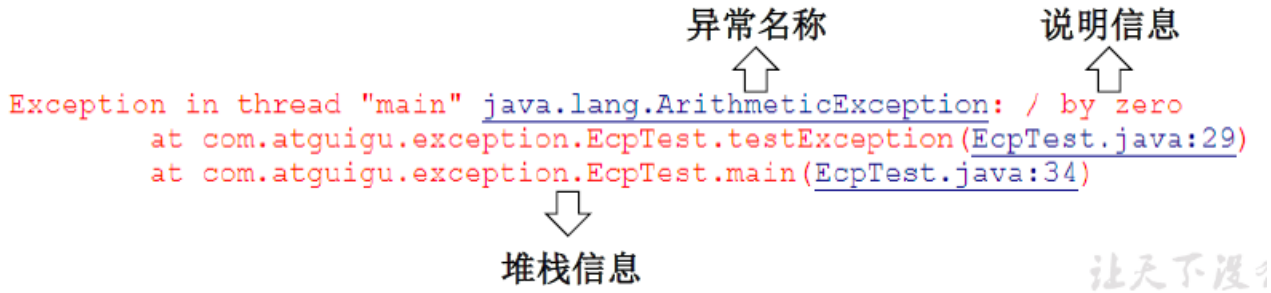
体会2：开发中，由于运行时异常比较常见，所以我们通常就不针对运行时异常编写try-catch-finally了。

针对于编译时异常，我们说一定要考虑异常的处理。

3.捕获异常的有关信息

与其它对象一样，可以访问一个异常对象的成员变量或调用它的方法。

- getMessage（）获取异常信息，返回字符串
- printStackTrace（）获取异常类名和异常信息，以及异常出现在程序中的位置。返回值void

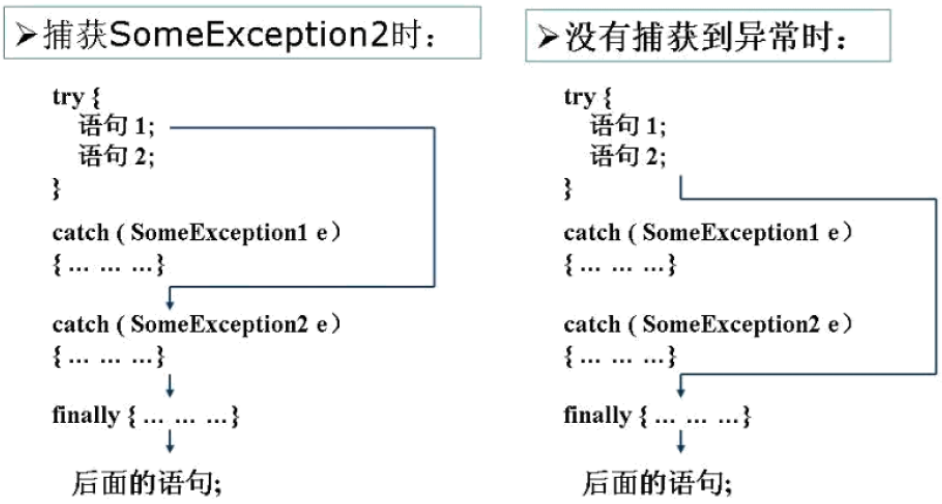


4.try-catch-finally中finally的使用

1.finally是可选的

2.finally中声明的是一定会被执行的代码。即使catch中又出现异常了，try中有return语句，catch中有return语句等情况。

3.像数据库连接、输入输出流、网络编程Socket等资源，JVM是不能自动的回收的，我们需要自己手动的进行资源的释放。此时的资源释放，就需要声明在finally中。




```java
public class ExceptionTest1 {
@Test
public void test2(){
try{
File file = new File("hello.txt");
FileInputStream fis = new FileInputStream(file);

int data = fis.read();
while(data != -1){
 System.out.print((char)data);
 data = fis.read();
 }

 fis.close();
 }catch(FileNotFoundException e){
 e.printStackTrace();
 }catch(IOException e){
 e.printStackTrace();
 }
 }

 @Test
 public void test1(){

 String str = "123";
 str = "abc";
 int num = 0;

 try{
 num = Integer.parseInt(str);

 System.out.println("hello-----1");
```

```
}catch(NumberFormatException e){
// System.out.println("出现数值转换异常了，不要着急....");
//String getMessage():
//System.out.println(e.getMessage());
//printStackTrace():
e.printStackTrace();
}catch(NullPointerException e){
System.out.println("出现空指针异常了，不要着急....");
}catch(Exception e){
System.out.println("出现异常了，不要着急....");

}
System.out.println(num);

System.out.println("hello-----2");
}
}
```

3、异常处理的方式二：throws + 异常类型

1. "throws + 异常类型"写在方法的声明处。指明此方法执行时，可能会抛出的异常类型。一旦当方法体执行时，出现异常，仍会在异常代码处生成一个异常类的对象，此对象满足throws后异常类型时，就会被抛出。异常代码后续的代码，就不再执行！

2. 体会：try-catch-finally:真正的将异常给处理掉了。throws的方式只是将异常抛给了方法的调用者。 并没有真正将异常处理掉。

3. 开发中如何选择使用try-catch-finally 还是使用throws？

3.1 如果父类中被重写的方法没有throws方式处理异常，则子类重写的方法也不能使用throws，意味着如果子类重写的方法中有异常，必须使用try-catch-finally方式处理。

3.2 执行的方法a中，先后又调用了另外的几个方法，这几个方法是递进关系执行的。我们建议这几个方法使用throws的方式进行处理。而执行的方法a可以考虑使用try-catch-finally方式进行处理。

```
public class ExceptionTest2 {

 public static void main(String[] args){
 try{
 method2();
 }catch(IOException e){
 e.printStackTrace();
 }
 }
  public static void method3(){
  try {
```

```
 method2();
 } catch (IOException e) {
 e.printStackTrace();
 }
 }

 public static void method2() throws IOException{
 method1();
 }

 public static void method1() throws FileNotFoundException,IOException{
 File file = new File("hello1.txt");
 FileInputStream fis = new FileInputStream(file);

 int data = fis.read();
 while(data != -1){
 System.out.print((char)data);
 data = fis.read();
 }
 fis.close();
 System.out.println("hahaha!");
 }
}
```

方法重写的规则之一：子类重写的方法抛出的异常类型不大于父类被重写的方法抛出的异常类型

```
public class OverrideTest {
 public static void main(String[] args) {
 OverrideTest test = new OverrideTest();
 test.display(new SubClass());
 }

 public void display(SuperClass s){
 try {
 s.method();
 } catch (IOException e) {
 e.printStackTrace();
 }
 }
}

class SuperClass{
 public void method() throws IOException{
```

```
 }
}

class SubClass extends SuperClass{
 public void method()throws FileNotFoundException{
 }
}
```