第6章 异常（重点）

部分内容摘自

《Java面向对象编程》，孙卫琴

《Java 程序设计》，唐大仕

1 为什么需要异常？ Exception

C 语言里面打开文件的典型代码

```
FILE *fp = fopen("D:\\demo.txt","rb");
if( fp== NULL ){
  printf("Error on open D:\\demo.txt file!");
  exit(1);
}
```

如果省略红色代码，编译时候**不会**报错，但是结果是**不可预料**的！！

下面情形怎么能让程序员知道所有潜在的危险？

```
public int getBalance(String accountNo);
//1账户不存在
//2网络连接故障
...
public boolean login(String username,String password)
//1用户名口令无效
//2用户被加入黑名单
//3用户登录可能存在盗号的可能
//...
```

常见的一些潜在风险

用户不存在？

用户名口令错误？

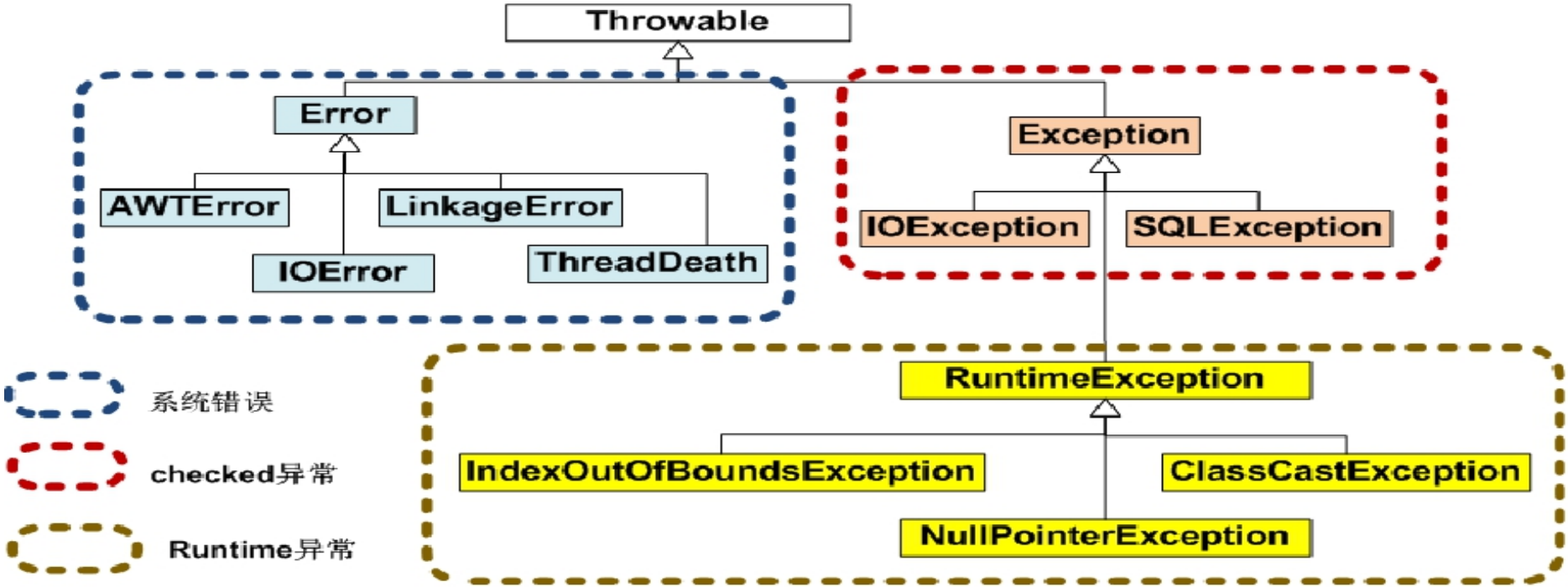
余额不足？

网络不通？

磁盘空间不足？

……

2 异常处理的基本模式-Java中的异常



异常处理基本模式1 ch06.ExceptionDemo

```java
public class ExceptionDemo {
    //一个可能抛出异常的方法(但是不一定抛出异常),
    public int getBalance(String username) throws Exception {
        //这个地方实际上应该访问数据库查询，为了简单，我们直接判断
        if(username.equals("张三")) {
            return 10000;
        }
        //抛出了一个异常
        throw new Exception("用户"+username+"不存在！");
    }

    public static void main(String[] args) {
        String name="李四";
        ExceptionDemo demo=new ExceptionDemo();
        try {
            //执行下面这句话 可能发生异常
            int balance=demo.getBalance(name);
            //如果发生了异常 下面这句话不会被执行
            System.out.println("The balance of "+name+" is:"+balance);
        }catch(Exception e) {
            //只有发生了异常下面这句话才会被执行
            System.out.println("出事了："+e.getMessage());
            //打印错误堆栈
            e.printStackTrace();
        }
    }
}
```

异常处理基本模式1

- 在Java编程语言中，用try和catch语句来处理异常。格式如下：

```
try {
    //可能发生异常的语句
} catch (Exception e) {
  //发生异常了 执行这里
}
```

异常处理基本模式 2 ch06.ExceptionDemo2

```
public class ExceptionDemo2 {
    public int getBalance(String username) throws Exception {
        // 这个地方实际上应该访问数据库查询，为了简单，我们直接判断
        if (username.equals("张三")) {
            return 10000;
        }
        throw new Exception("用户" + username + "不存在！");
    }
    public static void main(String[] args) {
        String name = "张三";
        ExceptionDemo demo = new ExceptionDemo();
        try {
            int balance = demo.getBalance(name);
            // 不发生异常时候下面代码会被执行
            System.out.println("The balance of " + name + " is:" + balance);
        } catch (Exception e) {
            // 发生异常的时候下面代码会被执行
            System.out.println("出事了：" + e.getMessage());
            // e.printStackTrace();
        } finally {
            // 不管是否发生异常，下面代码都会被执行
            System.out.println(name + " 试图访问账户余额");
        }
    }
}
```

异常处理基本模式

```
try {
  //可能发生异常的语句
    //如果前面发生了异常 就不会执行这里
 } catch (Exception e) {
   //发生异常了 执行这里 如果没有发生异常 不会执行这里
 }finally{
   //不管是否发生异常都会执行这里
 }
```

注意finally不管如何 都会执行这里

3 异常的产生

异常是怎么产生的?

所有的异常 都是通过throw 创建出来的

创建方法：

throw new 异常的构造函数()....

例如：throw new Exception("用户不存在！");

我能随便的抛出一个东西么？

不可以！

如下代码就是错误的：

throw new String ("用户不存在！");

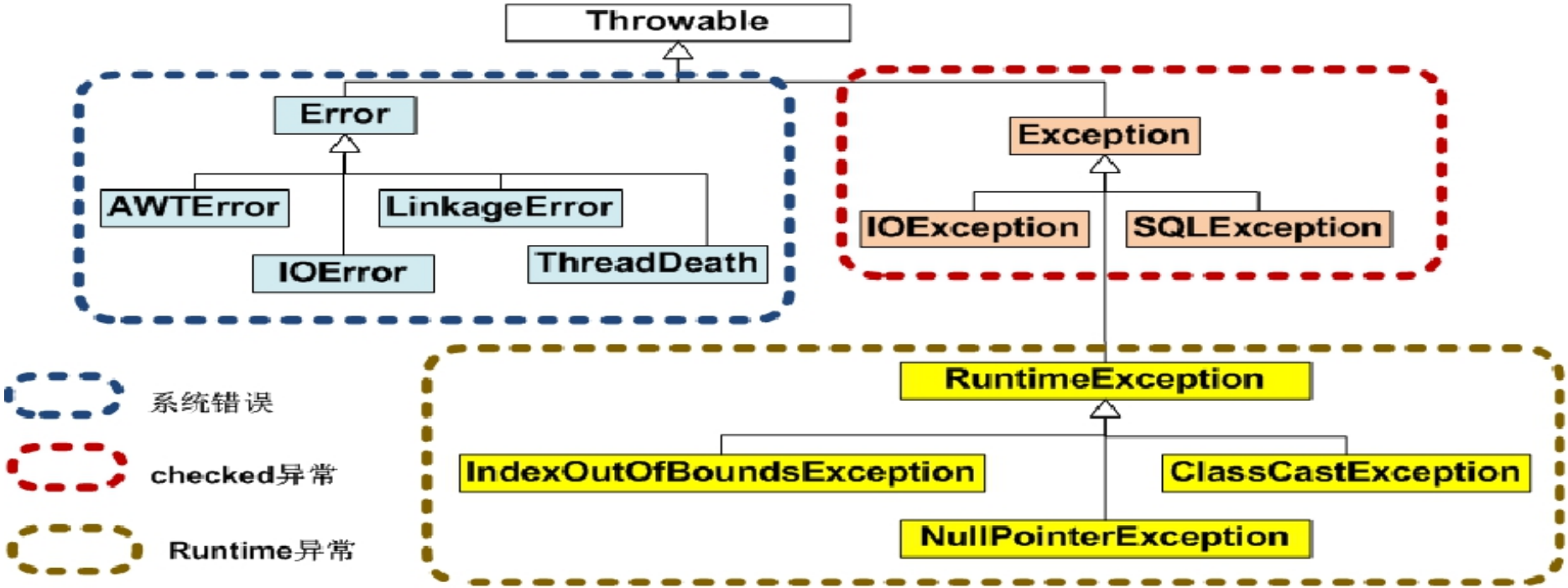
抛出的内容必须是**Throwable**的子类。

例子：ch06.ExceptionDemo3

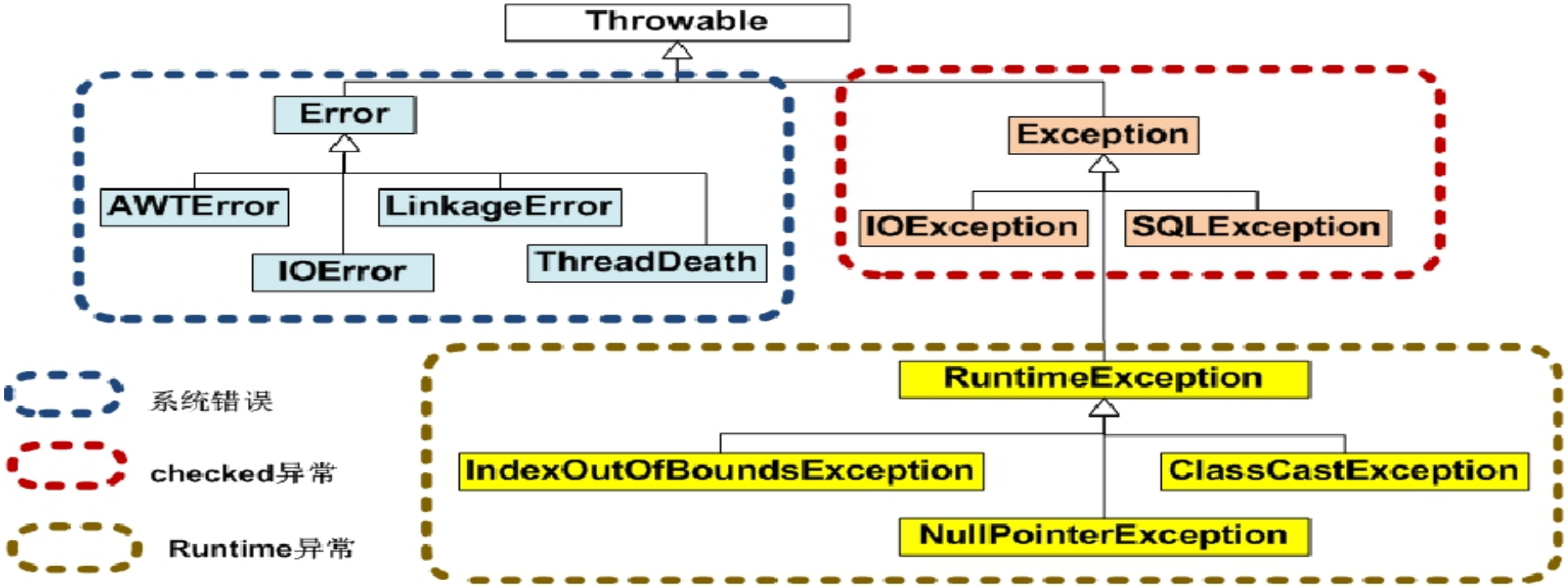
```java
public class ExceptionDemo3 {
    public int getBalance(String username) throws Exception {
        // 这个地方实际上应该访问数据库查询，为了简单，我们直接判断
        if (username.equals("张三")) {
            return 10000;
        }
        // 不是什么都能throw
        throw new String("用户" + username + "不存在！");
    }
    public static void main(String[] args) {
        String name = "张三";
        ExceptionDemo demo = new ExceptionDemo();
        try {
            int balance = demo.getBalance(name);
            System.out.println("The balance of " + name + " is:" + balance);
            // 不是什么都能catch
        } catch (String s) {
            System.out.println("出事了：" + e.getMessage());
            // e.printStackTrace();
        } finally {
            System.out.println(name + " 试图访问账户余额");
        }
    }
}
```

Java中什么能throw /catch?

:只要Throwable的子类，都可以



不是Throwable的子类，不能throw/catch



可抛出Throwable的几个类型

- Error
- Exception
- RuntimeException

Error及其子类 ch06.ErrorDemo

- Error类表示Java运行时产生的系统内部错误或资源耗尽等严重错误。
- 这种错误通常是程序无法控制和解决的，如果发生这种错误，通常的做法是通知用户并中止程序的执行。
- 这种异常超出了程序员能力范围，没解决办法，建议程序直接退出。编译器检查、必须处理。

例：NoClassDefFoundError、OutOfMemoryError、VirtualMachineError、InternalError StackOverflowError、UnknownError

Exception及其子类(不包括RuntimeException)

ch06.ExceptionDemo2

- 又称为“可检异常”“非运行时异常”
- 程序能处理的异常
- 这种错误的出现完全在程序员意料之中，有对应的解决办法 .编译器检查、必须处理。

IOException、SQLException…

很多用户自定义的：口令错误、余额不足...

RuntimeException及其子类

- RuntimeException类及其子类被称为“运行时异常”
  - 一般发生在JRE内部
  - 也称“非必检异常”
  - 如NullPointerException
- RuntimeException类及其子类都称为运行时异常，这种异常的特点是Java编译器不会检查它. 也就是说，当程序中可能出现这类异常，即使没有用try-catch语句捕获它，也没有用throws子句声明抛出它，也会编译通过。编译器不检查。

RuntimeException及其子类

ch06.RuntimeExceptionDemo

- **RuntimeException**表示无法让程序恢复运行的异常，导致这种异常的原因通常是由于执行了错误操作。一旦出现了错误操作，建议直接显示错误，因此Java编译器不检查这种异常；
- 也有一种说法认为**RuntimeException之所以不检测，是因为不应该替别人背黑锅！**

常见RuntimeException

- ArithmeticException

```
int a=12 / 0;
```

- NullPointerException

```
Date d= null;
System.out.println(d.toString());
```

- ArrayIndexOutOfBoundsException

```
int[] array=new int[4];
array[0]=1;
array[4]=1;
```

- ClassCastException

```
Animal animal=new Dog();
Cat cat=(Cat)animal;
```

创建用户自定义异常类

- 创建用户自定义异常时，一般需要完成如下的工作。
- (1) 声明一个新的异常类，使之以Exception类或其他某个已经存在的系统异常类或 用户异常类为父类。
- (2) 为新的异常类定义属性和方法，或重载父类的属性和方法，使这些属性和方法能够体现该类所对应的错误的信息。
- 例如：ch06. UserDefineException

遇到异常，该怎么办？异常的处理

- 1 自己处理 ch06. ExceptionDemo4

```
public int getNewBalance1(String username, int amount) {
    int orgBalance = 0;
    try {
        orgBalance = this.getBalance(username);
    } catch (Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }
    return orgBalance - amount;
}
```

- 2 自己不处理 继续throw

  ch06.ExceptionDemo5，等着别人处理

```
public int getNewBalance2(String username, int amount) throws Exception {
    int orgBalance = this.getBalance(username);
    return orgBalance - amount;

}
```

如何分门别类的处理异常

- 定义多种异常类(Throwable子类)，不同情况下，抛出不同异常类
- 例子：ch06. ExceptionDemo6

```
public static void main(String[] args) {
    String name = "张三o";
    ExceptionDemo6 demo = new ExceptionDemo6();
    try {
        int balance = demo.buy(name, 100000);
        System.out.println("购物成功！当前余额：" + balance);
    } catch (InvalidUserException e) {
        System.out.println("用户不存在：" + e.getMessage());
    } catch (BalanceException e) {
        System.out.println("没钱别买东西：" + e.getMessage());
    } finally {
        // 不管发生什么情况，都要执行如下代码
        System.out.println(name + " 试图访问账户余额");
    }
}
```

如果抛出了可以捕捉的东西，一定要在方法后面加throws 么？

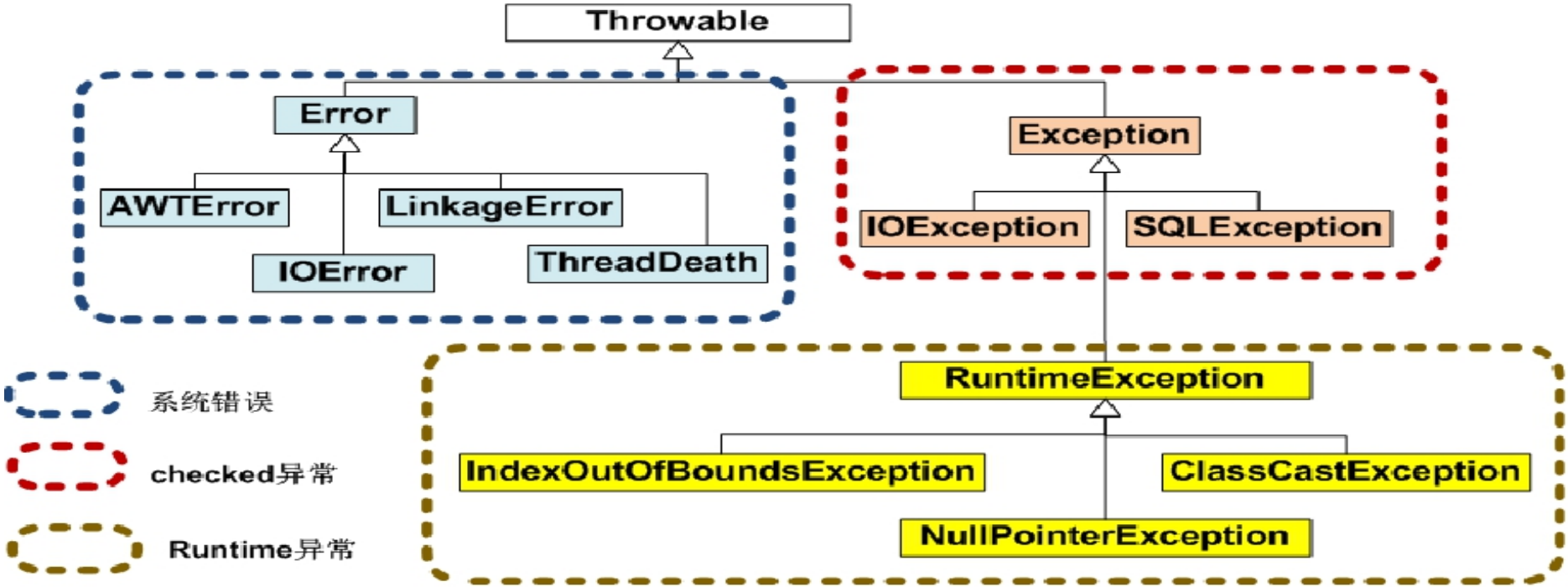
ch06.ExceptionDemo8

- 有时候需要

如果抛出的是Exception的子类，且非RuntimeException的子类，必须抛出或者捕捉

- 有时候不需要

如果抛出的是RuntimeException的子类，或者Error子类，则不需要显示的抛出


Throwable
Error
AWTError
LinkageError
IOError
ThreadDeath
Exception
IOException
SQLException
RuntimeException
IndexOutOfBoundsException
ClassCastException
NullPointerException
系统错误
checked异常
Runtime异常

可不可以在不抛出内容的情况下catch

- 有时候不行：ch06.ExceptionDemo9

Exception子类且 非RuntimeException的子类，必须在抛出的情况下捕捉。

- 有时候行： ch06.ExceptionDemo10

RuntimeException的子类或者Error的子类，可以在不抛出的情况下捕捉。

catch Exception 能捕捉到所有抛出的“东西” 么?

- ch06.ExceptionDemo10
- 不能，要写捕捉所有抛出的“东西”，必须 catch Throwable

finally语句 是被执行的代码块，而不管有没有出现异常。

```
public void work() {
  try{
   开门
   工作8个小时
//可能会抛出DiseaseException异常
   关门
  }catch(DiseaseException e){
   去医院看病;
  }
}
```

```
public void work() {
  try{
   开门
   工作8个小时
//可能会抛出DiseaseException异常
  }catch(DiseaseException e){
    去医院看病;
  }finally{
    关门
  }
}
```

异常的优点

- 极大的降低了程序调试难度，使得错误定位准确
- 强制让用户在编译阶段发现一些未解决问题，提高程序健壮性

异常引发的一系列问题

1 finally一定被执行么？

不一定，finally语句不被执行的唯一情况是程序先执行了终止程序的System.exit()方法

ch06. ExceptionDemoExit

ch06.FinallyDemo

```
public static void test() {
    try {
        return ;
    }catch(Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }finally {
        //这行代码会被执行么？
        System.out.println("finally");
    }
}
```

2 异常的顺序问题：异常会首先被符合条件的catch捕捉

```
try {
    throw new java.io.IOException("ie");
    //IOException是Exception的子类
} catch (IOException ie) {
    //异常在这里被捕捉了
    System.out.println("IOException");
} catch (Exception e) {
   //这段不会执行了
    System.out.println("Exception");
}
```

```
try {
    throw new java.io.IOException("ie");
} catch (IOException ie) {
 System.out.println("IOException");
} catch (Exception e) {
 System.out.println("Exception");
}//正确

try {
    throw new java.io.IOException("ie");
} catch (Exception e) {
  System.out.println("Exception");
} catch (IOException  ie) {
   System.out.println(“IOException");
}//编译错误，为什么？
```

3 返回问题 下面函数为什么报错

ch06. ExceptionDemo20.buy

```
public int buy(String username, int amount) throws Exception {
    try {
        int orgBalance = this.getBalance(username);
        if (orgBalance < amount) {
            throw new BalanceException("余额不足！");
        }
        return orgBalance-amount;
    } catch (Exception e) {
        System.out.println(e.getMessage());
        //throw e;
        //e.printStackTrace();
        //throw new Exception(e.getMessage());
    }
}
```

3 初始化问题 下面函数为什么报错

ch06. ExceptionDemo20.buy2

```
public void buy2(String username, int amount) throws Exception {
    int orgBalance;
    try {
        orgBalance = this.getBalance(username);
        if (orgBalance < amount) {
            throw new BalanceException("余额不足！");
        }
    } catch (Exception e) {
        System.out.println("余额不足:"+orgBalance);
    }
}
```

3 初始化问题 下面函数为什么报错

ch06.ExceptionDemo20. readFromFile

```
public void readFromFile(String fileName) {
    InputStream in;
    try {
        in = new FileInputStream(fileName);
        //....以后还有代码，此处省略
    } catch (FileNotFoundException e) {
        // TODO Auto-generated catch block
        e.printStackTrace();
    } finally {
        if(in!=null) {
            in.close();
        }
    }
}
```

4 Exception引发的override问题

ch06. ExceptionSon

- 一个方法必须通过throws语句在方法的声明部分说明它可能抛出而并未捕获的所有的“必检异常”，如果没有这么做，将不能通过编译。
- 如果在子类中覆盖了父类的某一方法，那么该子类方法不可以比被其覆盖的父类方法抛出更多的异常（但可以更少）。

5 异常的常见处理方法：

ch06. ExceptionDemo31

```
public static void main(String[] args) {
    try {
        int[] a=new int[10];
        for(int i=0;i<=a.length;i++) {
            a[i]=i;
        }
    }catch(Exception e) {
        System.out.println(e.getMessage());
        e.printStackTrace();
    }

}
```

6 处理异常的过程中，可能再次发生异常

ch06. ExceptionInException

throw与throws 用处的区别ch06. ThrowsDemo

```
public void method()
throws SQLException, IOException {
  if(1>2) {
    throw new IOException("IOException");
  }
  if(2>3) {
    throw new SQLException("SQLException");
 }
}
```

7 异常最大的用途-查看错误堆栈

ch06. ExceptionDemo31

e.printStackTrace();

java.lang.ArrayIndexOutOfBoundsException: 10
at ch06.ExceptionDemo31.main(ExceptionDemo31.java:14)

Exception 包含在java.lang包

所以使用 Exception不需要import

8 异常到底应该捕捉还是抛出？

- 捕获并处理那些你知道并且应该负责处理的异常
- 对那些你不知道方法的调用者会如何处理的异常，最好将它们留给方法的调用者进行处理。