1. 异常处理
2. 异常出现后

当异常出现后，我们希望程序能够做到以下几点:

1. 向用户通知错误；
2. 保存所有的工作；
3. 允许用户妥善地退出程序；

异常处理的任务就是将控制权从产生错误的地方转移到能够处理这种情况的错误处理器中。

1. 异常的分类

Java中，所有的异常对象都是派生于Throwable类的一个类实例，这里说的异常不止通常所指的Exception，还有Error这种错误。Error类描述了Java运行时系统的内部错误和资源耗尽错误，如果出现这种错误，除了通知用户和尽力地妥善终止程序外，我们是无能为力的。而对于Exception类，又分为RuntimeException和其他异常。

1. RuntimeException

这种异常一般是由编程的错误导致的，如数组访问越界，错误的强制类型转换等等，通俗地讲，只要是RuntimeException异常，那么一定是你在编码时出现了问题。Java将派生于Error类或者RuntimeException类的所有异常称为非检查型异常。

1. 其他异常

所谓的其他异常指的就是非RuntimeException，这种异常不是由你的编码造成的，而是由编码的环境，如硬件，软件，网络等等造成的，且你是无法控制的，因此我们通常需要将这种异常通知编译器，让编译器做好准备。这种异常称为检查性异常，可以被抛出，抛出就代表着生命这个方法可能会出现这种异常，让方法的调用者来处理这个异常或者继续传递异常；也可以在代码中捕获异常并进行异常处理。

一个方法必须声明所有可能抛出的检查型异常，不然，编译器就会发出一个错误信息。

注意，如果在子类中覆盖了超类的一个方法，子类方法中声明的检查型异常不能比超类方法中声明的异常更加通用。通用是啥意思呢？就是说子类方法只能抛出更加特定的异常，或者根本不抛出异常，如果子类方法抛出了超类中没有抛出的异常，那么就会出错。

1. 抛出异常

抛出异常时可以选择抛出默认的异常或是抛出一个带字符串参数的异常，后面一种可以详细地描述异常，只需要创建一个异常类对象，然后将该对象抛出即可，一旦方法抛出了异常，这个方法就不会返回到调用者。

throw new EOFException();

String gripe = “aa”;

throw new EOFException(grip);

1. 创建自己的异常类

自己的异常类需要派生于Exception类，或者派生于Exception的某个子类，如IOException，该异常类一般需要包含两个构造器，一个是默认的构造器，一个是包含详细描述信息的构造器，超类Throwable的toString方法会返回一个字符串，其中包含这个的详细信息。如下所示：

class FileFormatException extends IOException{

public FileFormatException(){}

public FileFormatException(String gripe){

super(gripe);

}

1. 捕获异常
2. 捕获异常

如果发生了某个异常，但没有在任何地方捕获这个异常，那么程序就会终止，并在控制台上打印一个消息，包括这个异常的类型和一个堆栈轨迹。要想捕获异常，需要使用try/catch/finally语句块，try语句块中是正常的逻辑处理，也是可能会出现异常的位置；catch语句块中是捕获异常，并且处理异常的位置；finally代码块不论是否有异常都会执行，通常用于程序资源的释放。

1. 捕获异常的规则

通常，如果我们调用了一个抛出检查型异常的方法，就必须处理这个异常或者继续传递这个异常。一般的经验是可以捕获那些知道怎么处理的异常，继续传播那些不会处理的异常。

1. 再次抛出异常与异常链

可以在catch语句块中抛出一个异常，通常，如果我们想要改变异常的类型时会这样做。最好的做法是将原始异常设置为新异常的“原因”，这样，当调用者捕获到新异常时，可以通过新异常获取到原始的异常，这种包装技术可以让我们在子系统中抛出高层异常，而不会丢失原始的异常细节。如下所示：

try{

}

catch(SQLException original){

var e = new ServletException(“database error”);

e.initCause(original);

throw e;

}

在捕获到新异常时，通过getCause()方法获取原始异常：

Throwable original = caughtException.getCause();

1. finally子句

finnaly子句一般用于处理资源的释放，不要把改变控制流的语句如return , throw ,break , continue放在finally子句中，这样可能造成意想不到的结果。如finally子句中有return 语句，那么方法返回前，会执行finally子句块中的return语句，这个返回值将会遮蔽原有的返回值。

1. try-with-resources语句

使用try-with-resources语句可以有效地简化资源的释放，在try语句块的参数中将会使用的资源传入，它会自动地释放这个资源，不需要再finally块中手动释放，使用如下所示：

try(var in = new Scanner(new FileInputStream(“/usr/share/dict/words”), StandardCharsets.UTF\_8)){

while(in.hasNext())

System.out.println(in.next);

}