1 处理错误

* 1. 异常分类



Error类层次描述了Java运行时系统的内部错误和资源耗尽错误。

设计Java程序时，主要关注Exception层次结构。

由程序错误导致的异常属于RuntimeException ，而程序本身没有问题，但由于像I/O错误这类问题导致的异常属于其他异常。

RuntimeException包含下面几种情况：

* 错误的类型转换 ClassCastException
* 数组访问越界 ArrayIndexOutOfBoundsException
* 访问空指针 NullPointerException
* 指定的类不存在ClassNotFoundException
* 数学运算符异常 ArithmeticException
* 方法参数错误 IllegalArgumentException
* 没有权限访问 IllegalAccessException

不是派生域RuntimeException的异常包括：

* 试图在文件尾部后面读取数据
* 试图代开一个不存在的文件。
* 试图根据给定的字符串查找Class对象，而这个字符串表示的类并不存在。
  1. 声明已检查异常

如果无法处理，抛出一个异常。

例如：一段读取文件的代码知道有可能读取的文件不存在，或者内容为空，因此试图处理文件信息的代码就需要通知编译器可能会抛出IOException类的异常。

在方法处 throws XXXException

* 1. 如何抛出异常

方法体内

throw new XXXException();

* 1. 创建异常类

定义一个派生于Exception或Exception子类的类

定义的类应该包含两个构造器

一个是默认构造器，另一个是带有详细描述信息的构造器

1. 捕获异常

try catch

通常 应该捕获那些知道如何处理的异常，而将那些不知道怎样处理的异常继续进行传递。

try catch catch 。。。 捕获多个异常

catch中再次抛出异常

finally

不管是否有异常被捕获，finally子句中的代码都被执行。 关闭文件读入流，关闭资源时用到

1. 使用异常机制的技巧

* 异常处理不能代替简单的测试 使用if判断
* 不要过分地细化异常
* 利用异常层次结构
* 不要压制异常
* 在检测错误，苛刻要比放任好
* 不要羞于传递异常

1. 使用断言

断言机制允许在测试期间向代码中插入一些检查语句。当代吗发布时，这些插入的检测语句会自动地移走。

assert 条件;

assert 条件: 表达式;

这两种形式都会对条件进行检测，如果结果为false，则抛出一个AssertionError异常。在第二种形式中，表达式将被传入AssertionError的构造器，并转换成一个消息字符串。

* 1. 启用和禁用断言

在默认情况下，断言被禁用。可以运行程序时用-enableassertions或-ea选项启用它

java –ea XXapp

注意:在启用或禁用断言时不必重新编译程序。启用或禁用断言是类加载器的功能。当断言被禁用时，类加载器将跳过断言代码，因此不会降低程序运行的速度。

也可以在某个类或某个包中使用断言

* 1. 使用断言完成参数检查

在Java语言中，给出了3种处理系统错误的机制：

抛出一个异常

日志

使用断言

什么时候选择断言：

断言失败是致命的，不可恢复的错误，

断言检查只用于开发和测试阶段

@param a the array to be sorted （must not be null）

assert a != null;

* 1. 为文档假设使用断言

1. 记录日志

记录日志API优点：

1可以很容易地取消全部日志记录，或者仅仅取消某个级别的日志，而且打开和关闭这个操作也很容易

2可以很简单地禁止日志记录的输出，因此，将这些日志代码留在程序中的开销最小。

3 日志记录可以被定向到不同的处理器，用于在控制台中显示，用于存储在文件中等

4 日志记录器和处理器都可以对记录进行过滤。过滤器可以根据过滤实现器制定的标准丢弃那些无用的记录项。

5 日志记录可以采用不同的方式格式化，例如纯文本和xml

6 应用程序可以使用多个日志记录器，它们使用类似包名的这种具有层次结构的名字

7 在默认情况下，日志系统的配置由配置文件控制

* 1. 基本日志

Logger.global 默认记录器，可以用System.out替换它，并通过调用info方法记录日志信息。

Logger.getGlobal().info(“ ”);

PS:自动包含了时间，调用的类名和方法名，但是在相应的地方调用：

Logger.getGlobal().setLevel(Level.OFF); 将会取消所有的日志

* 1. 高级日志

企业级日志，不要将所有的日志都记录到一个全局日志记录器中，而是自定义日志记录器

调用getLogger方法可以创建或检索记录器。

与包名类似，日志记录器名也具有层次结构。

7个日志记录级别

SEVERE

WARING

INFO

CONFIG

FINE

FINER

FINEST

在默认情况下，只记录前三个级别，或者使用Level.OFF关闭所有级别的记录。

Logger.setLevel(Level.FINE);

FINE和更高级别的记录都可以记录下来。

可以使用Level.ALL开启所有级别的记录

记录方法：

logger.waring(message);

logger.fine(message);

logger.log(Level.FINE,message);

默认的日志记录将显示包含日志调用的类名和方法名，如同堆栈所显示的那样。但是，如果虚拟机对执行过程进行了优化，就得不到准确的调用信息。此时，可以调用logp方法获得调用类和方法的确切位置：

void logp(Level L ,String className;String methodName,String message)

* 1. 修改日志管理器配置

可以通过编辑配置文件来修改日志系统的各种属性。e/lib/loggin.properties

要想使用另外一个配置文件，就要将java.util.logging.config.file特性设置为配置文件的存储位置，并用下列命令启动应用程序

java –Djava.util.logging.config.file-config-File mainClass;

.level=INFO

* 1. 本地化

本地化的应用程序包含资源包中的本地特定信息。

* 1. 处理器

日志记录器将记录发送到ConsoleHandler中，并由它输出到System.err流中。

默认情况下，日志记录器将记录发送到自己的处理器和父处理器。要将日志记录发送到其他地方，就要添加其他的处理器。日志API提供了FileHandler，另一个是SocketHandler。SocketHandler将记录发送到特定的主机和端口，FileHandler可以收集文件中的记录。

可以像下面这样直接将记录发送到默认文件的处理器

FileHandler handler = new FileHandler();

logger.addHandler(handler);

这些记录被发送到用户主目录的javan.log文件中，n是文件的唯一编号。

还可以通过扩展Handler或StreamHandler类自定义处理器。

* 1. 过滤器

根据日志记录的级别进行过滤

也可以通过实现Filter接口并定义下列方法来自定义过滤器

boolean isLoggable(LogRecord record)

* 1. 格式化器

ConsoleHandler类和FileHandler类可以生成文本和XML格式的日志记录。

也可以自定义格式，需要扩展Formater类并覆盖下面这个方法

String format(LogRecord record)

* 1. 日志记录说明

1 为一个简单的应用程序，选择一个日志记录器，并把日志记录器命名为与应用程序包一样的名字。

2 默认的 日志配置将级别等于或高于INFO级别的所有消息记录到控制台。用户可以覆盖默认的配置文件。

3 级别为INFO、WARING、SEVERE的消息将显示在控制台上，将有意义的消息设置为这几个级别。

1. 调试技巧

1 使用打印任意值

2 类中放一个main方法 单元测试

3 Junit

4 日志代理

5 利用Throwable类提供的printStackTrace方法，可以从任何一个异常中获得堆栈情况。

6 堆栈跟踪显示在System.err上

7 将一个程序中的错误信息保存在一个文件中

8 让非捕获异常的堆栈跟踪出现在System.err中并不是一个很理想的方法。

9 可以用-verbose启动java虚拟机观察类的加载过程

10 Xlint 选项编译器对一些普通容易出现的代码问题进行检查

11 java虚拟机对java应用程序进行监控和管理的支持

12可以使用jmap实用工具获得一个堆的转储，其中显示了堆中的每个对象。

13 如果使用-Xprof标志运行java虚拟机，就会运行一个基本的剖析器来跟踪那些代码中经常被调用的方法。

1. 使用调试器

在Debug中

Step Into 命令跟踪到每个方法调用的内部 Eclipse F5

Step Over 命令定位到下一行，而并不跟踪到方法调用的内部 Eclipse F6