1. 断言
2. 概述

断言机制允许在测试期间向代码中插入一些检查，而在生产过程中会自动删除这些检查。如果我们不确定某些属性是否符合要求，就可以使用断言来检查。

1. assert关键字

assert关键字用于断言，有两种形式，一种是只添加断言要判断的条件，如下所示：

assert condition;

另一种是不仅计算条件，还添加一条表达式，当断言的结果为false时，就会抛出一个AssertionError异常，在后面的表达式中，表达式将传入AssertionError对象的构造器，并转换为一个消息字符串。如下所示：

assert condition : expression

表达式的唯一目的是产生一个消息字符串，用于在断言失败时显示。

1. 启用断言
2. 命令行中

默认情况下断言是禁用的，可以在CMD中启用断言，通过-enableassertions 或者 -ea选项来启用断言，如下所示, MyApp是要启用断言的方法：

java -enableassertions MyApp

也可以为某个类或者整个包启用断言，如启用MyClass类的断言：

java -ea:MyClass

启用mylib包以及子包的断言：

java -ea:com.company:mylib

注意！！！不需要重新编译来启用或者禁用断言，启用或禁用断言是类加载器的功能，禁用断言时，类加载器会去除断言代码，因此不会降低程序运行的速度。

1. IDEA

在当前项目的 Edit Configuration下，在 VM Option中填写 -enableassertions 或者 -ea， Apply后即可开启断言。

1. 禁用断言

在启用断言后，要想禁用断言的话，通过-disableassertions或者-da在某个特定类或者包中禁用断言即可。

java -da:MyClass

1. 系统类的断言

启用和禁用断言的 -ea 和 -da开关不能应用到那些没有类加载器的系统类中，这些系统类需要使用 -enablesystemassertions 或者 -esa 开关来开启断言

1. 什么时候使用断言

断言失败是致命的，不可恢复的错误，不应该使用断言向程序的其他部分通知发生了可恢复性的错误，断言只应该用于在测试阶段确定程序内部错误的位置。当我们需要使用到一个条件时，可以首先判断该条件是否是真，使用断言就可以保证程序运行时该条件的正确性。

1. 日志
2. 日志的优点
3. 可以设置日志的级别，记录对应级别的日志。
4. 日志可以被定向到不同的处理器中，如控制台显示，文件读写等等。
5. 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器实现器指定的标准丢弃无用的记录项。
6. 日志记录可以采用不同的方式格式化，如纯文本或者XML。
7. 基本日志

使用全局日志记录器的info方法可以记录info中的参数字符串。

Logger.*getGlobal*().info("File->Open menu item selected");

也可以通过设置日志等级来取消所有的日志：

Logger.*getGlobal*().setLevel(Level.*ALL*);

1. 高级日志
2. 定义自己的日志记录器

通过getLogger方法，方法的参数为该日志记录器的名称：

private static final Logger *myLogger* = Logger.*getLogger*("com.mycompany.myapp");

注意，这里需要使用静态变量存储日志记录器的引用，因为未被任何变量引用的日志记录器可能会被垃圾回收，而日志记录器可能需要一直存在着，所以需要用静态变量。

1. 日志记录器名的层级关系

日志记录器名也是有层级结构的，父与子之间将共享某些属性，如果对父日志记录器设置了日志级别，那么它的子日志记录器也会继承这个级别。他们之间的继承关系是通过日志记录器名体现的，跟包名一样，”com.company.son” 是 “com.company”的子日志记录器。

1. 日志级别

总共有7个日志级别：

SEVERE

WARNING

INFO

CONFIG

FINE

FINER

FINEST

默认情况下，只记录INFO及以上的三个级别，当然我们可以修改设置一个不同的级别：

*myLogger*.setLevel(Level.*WARNING*);

这样，该日志记录器就会只记录WARNING及以上的日志了。

还可以使用Level.ALL开启所有级别的日志记录，使用Level.OFF关闭所有级别的日志记录。

Logger.getGlobal().setLevel(Level.ALL);

Logger.getGlobal().setLevel(Level.OFF);

1. 日志记录方法

除了ERROR级别，其他的日志级别都有日志记录方法，如下所示：

logger.warning(message);

同时，还可以使用log方法记录日志并且指定日志级别：

*myLogger*.log(Level.*WARNING*, "close now");

但是，这个方法依然不能指定ERROR级别的日志，应该是ERROR级别的日志不能随意指定，应该由系统在发生错误时进行记录。

1. 显示调用堆栈

默认的日志记录将显示根据调用堆栈得出的包含日志调用的类名和方法名，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，可以使用logp方法获得调用类和方法的准确位置。

void logp(Level l, String className, String methodName, String message)

可以看到，参数为日志界别，类名，方法名，以及日志记录信息。

1. 记录日志的用途

记录日志通常用于记录哪些意料之外的异常，可以在catch语句中捕获异常，并且将异常记录到日志之中。

1. 日志管理器配置
2. 配置文件

可以通过编辑配置文件来修改日志系统的各个属性，默认的配置文件为当前使用的JDK的conf/logging.properties文件。如果想要使用其他的配置文件，就要在启动程序时设置java.util.logging.config.file属性为那个配置文件的位置：

java -Djava.util.logging.config.file=…

1. 修改默认的日志级别

编辑配置文件的如下命令行，将日志级别修改为其他的：

.level=INFO

1. 指定自定义日志记录器的日志级别

在日志记录器名后添加后缀.level：

com.mycompany.myapp.level=FINE

1. 处理器级别设置

日志记录器仅仅记录日志，操作日志是由日志处理器来实施的。刚刚修改了自定义日志记录器的日志级别，要想该FINE级别日志输出到控制台上，就需要修改日志处理器的级别，同样是在配置文件中设置：

java.util.logging.ConsoleHandler.level=FINE

1. 日志管理器初始化

日志管理器在虚拟机启动时初始化，也就是在main方法执行前，如果没有使用命令初始化日志管理器，也可以在程序中使用如下语句，设置属性，并且初始化日志管理器：

System.setProperty(“java.util.logging.config.file”, file);

LogManager.getLogManager().readConfiguraion();

在Java9中，使用以下方式更新日志配置：

LogManager.getLogManager().updateConfiguraion(mapper);

这样，就会从java.util.logging.config.file系统属性指定的位置读取一个新的配置，然后使用mapper这个映射器来解析老配置或新配置中的所有键的值。

1. 日志处理器
2. 默认情况

日志记录器会将记录发送到自己的处理器和父处理器，最终的祖先处理器（名为“”）有一个ConsoleHandler，它会把所有高于默认级别INFO的记录输出到System.err流。

1. 日志处理器级别

日志管理器配置文件中将默认的控制台处理器的日志级别设置为，可以自定修改：  
java.util.logging.ConsoleHandler.level=INFO

1. 创建自己的处理器

创建一个处理器，并且将处理器添加到某个日志记录器上：

import java.io.IOException;  
import java.util.logging.ConsoleHandler;  
import java.util.logging.Level;  
import java.util.logging.Logger;  
  
public class loggerTest {  
  
 private static final Logger *myLogger* = Logger.*getLogger*("com.mycompany.myapp");  
  
 public static void main(String[] args) {  
 //Logger.getGlobal().setLevel(Level.ALL);  
 //Logger.getGlobal().info("File->Open menu item selected");  
 *myLogger*.setLevel(Level.*WARNING*);  
 *myLogger*.info("File->Open menu item selected");  
 *myLogger*.fine("File->Open menu item selected");  
 *myLogger*.warning("File->Open menu item selected");  
 //myLogger.log(Level.WARNING, "close now");  
 //myLogger.logp(Level.WARNING, "loggerTest", "main", "test");  
 //myLogger.log(Level.WARNING, "read");  
 ConsoleHandler consoleHandler = new ConsoleHandler();  
 consoleHandler.setLevel(Level.*FINE*);  
 *myLogger*.addHandler(consoleHandler);  
 }  
}

1. 其他的处理器
2. SocketHandler

将记录发送到指定的主机和端口。

1. FileHandler

将记录写到文件中，默认的文件处理器将记录发送到用户主目录的javan.log文件中，n是防止冲突的编号。如C://mrwxx目录下的 java1.log, java2.log，这些日志文件已经格式化为了XML文件，且可以通过设置日志管理器配置文件中的不同参数，修改文件处理器的默认行为。

如，向同一个日志文件中添加日志记录，需要设置如下参数：

java.util.logging.FileHandler.append=true

1. 日志过滤器

默认情况下，会根据日志记录的级别进行过滤，同时，每个日志记录器和处理器都有一个可选的过滤器来实现附加的过滤。要创建一个过滤器，就要实现Filter接口并且定义以下方法：

boolean isLoggable(LogRecord record);

在这个方法中，可以用自己的标准来分析日志，并且决定是否过滤。调用setFilter()方法就可以将一个过滤器安装到一个日志记录器或者处理器中，注意，同一时刻只能有一个过滤器。

1. 格式化器

控制器处理器和文件处理器可以生成文本和XML格式的日志记录，同时，我们也可以自己定义格式，这就需要创建一个格式类，继承Formatter类并且重写如下方法：

String format(LogRecord record)

1. 日志技巧
2. 可以将日志记录器命名为主应用包的名称。
3. 默认的INFO及以上的消息是对用户有意义的消息，而对于程序员来说，将日志级别设置为FINE是一个很好的选择。
4. 当我们在想要通过System.ou.println()方法调试时，可以尝试使用控制台处理器来记录日志信息，如下所示：

logger.fine(“File open failed”);

1. 在异常处理中，用日志记录下当前出现的异常以及想要记录的问题。