java断言

断言用于在开发和测试期间验证程序的正确性、并且在测试完成后不必保留他们，Java断言可以选择性开启。启用或禁用断言是类加载器的功能。

在防御式编程中经常会用断言（Assertion）对参数和环境做出判断，避免程序因不当的输入或错误的环境而产生逻辑异常在Java 中的断言使用的是assert 关键字，其基本的用法如下：  
    assert < 布尔表达式>  
    assert < 布尔表达式> : < 错误信息>  
 在布尔表达式为假时，抛出AssertionError 错误，并附带了错误信息。assert 的语法较简单，有以下两个特性：

* assert 默认是不启用的  
  我们知道断言是为调试程序服务的，目的是为了能够快速、方便地检查到程序异常，但Java 在默认条件下是不启用的，要启用就需要在编译、运行时加上相关的关键字，这就不多说，有需要的话可以参考一下Java 规范。
* assert 抛出的异常AssertionError 是继承自Error 的  
  断言失败后，JVM 会抛出一个AssertionError 错误，它继承自Error，注意，这是一个错误，是不可恢复的，也就表示这是一个严重问题，开发者必须予以关注并解决之。

assert 虽然是做断言的，但不能将其等价于if…else…这样的条件判断，它在以下两种情况不可使用：  
**（1）在对外公开的方法中**  
 我们知道防御式编程最核心的一点就是：所有的外部因素（输入参数、环境变量、上下文）都是“邪恶”的，都存在着企图摧毁程序的罪恶本源，为了抵制它，我们要在程序中处处检验，满地设卡，不满足条件就不再执行后续程序，以保护主程序的正确性，处处设卡没问题，但就是不能用断言做输入校验，特别是公开方法。我们来看一个例子：  
public class Client {  
    public static void main(String[] args) {  
        StringUtils.encode(null);  
    }  
}  
// 字符串处理工具类  
class StringUtils{  
    public static String encode(String str){  
        assert str!=null:" 加密的字符串为null";  
        /\* 加密处理\*/  
    }  
}  
 encode 方法对输入参数做了不为空的假设，如果为空，则抛出AssertionError 错误，但这段程序存在一个严重的问题，encode 是一个public 方法，这标志着是它对外公开的，任何一个类只要能够传递一个String 类型的参数（遵守契约）就可以调用，但是Client 类按照规范和契约调用enocde 方法，却获得了一个AssertionError 错误信息，是谁破坏了契约协定？—是encode 方法自己。  
  
**（2）在执行逻辑代码的情况下**  
 assert 的支持是可选的，在开发时可以让它运行，但在生产系统中则不需要其运行了（以便提高性能），因此在assert 的布尔表达式中不能执行逻辑代码，否则会因为环境不同而产生不同的逻辑，例如：  
public void doSomething(List list,Object element){  
    assert list.remove(element):" 删除元素 " + element + " 失败";  
    /\* 业务处理\*/  
}  
 这段代码在assert 启用的环境下，没有任何问题，但是一旦投入到生产环境，就不会启用断言了，而这个方法也就彻底完蛋了，list 的删除动作永远都不会执行，所以也就永远不会报错或异常，因为根本就没有执行嘛！  
  
 以上两种情况下不能使用assert，那在什么情况下能够使用assert 呢？一句话：按照正常执行逻辑不可能到达的代码区域可以放置assert。具体分为三种情况：

* **在私有方法中放置assert 作为输入参数的校验**  
  在私有方法中可以放置assert 校验输入参数，因为私有方法的使用者是作者自己，私有方法的调用者和被调用者之间是一种弱契约关系，或者说没有契约关系，其间的约束是依靠作者自己控制的，因此加上assert 可以更好地预防自己犯错，或者无意的程序犯错。
* **流程控制中不可能达到的区域**  
  这类似于JUnit 的fail 方法，其标志性的意义就是：程序执行到这里就是错误的，例如：  
  public void doSomething(){  
      int i = 7;  
      while(i >7){  
          /\* 业务处理\*/  
      }  
      assert false:" 到达这里就表示错误";  
  }
* **建立程序探针**

我们可能会在一段程序中定义两个变量，分别代表两个不同的业务含义，但是两者有固定的关系，例如var1=var2\*2，那我们就可以在程序中到处设“桩”，断言这两者的关系，如果不满足即表明程序已经出现了异常，业务也就没有必要运行下去了。