

第一章

简介:

jtest 是 parasoft 公司推出的一款针对 java 语言的自动化白盒测试工具,它通过自动实现 java 的单元测试和代码标准校验,来提高代码的可靠性。Jtest 先分析每个 java 类,然后自动生成 junit 测试用例并执行用例,从而实现代码的最大覆盖,并将代码运行时未处理的异常暴露出来;另外,它还可以检查以 DbC

(Design by Contract) 规范开发的代码的正确性。用户还可以通过扩展测试用例的自动生成器来添加更多的 junit 用例。Jtest 还能按照现有的超过 350 个编码标准来检查并自动纠正大多数常见的编码规则上的偏差,用户可自定义这些标准,通过简单的几个点击,就能预防类似于未处理异常、函数错误、内存泄漏、性能问题、安全隐患这样的代码问题。

另外, jtest 采用 parasoft 公司的 AEP 方法论来实现团队开发中对代码错误标准化管理,这个方法论,也是 parasoft 提出的基于团队开发模式下提高软件质量和可靠性的一套解决方案,目前还处在探索阶段,详情可参阅

<http://www.parasoft.com/jsp/aep/aep.jsp>。

好处:

- 1) 使预防代码错误成为可能,从而大大节约成本,提高软件质量和开发效率
- 2) 使单元测试——包括白盒、黑盒以及回归测试成为可能
- 3) 使代码规范检查和自动纠正成为可能
- 4) 鼓励开发团队横向协作来预防代码错误

特征:

- 1) 通过简单的点击,自动实现代码基本错误的预防,这包括单元测试和代码规范的检查
- 2) 生成并执行 junit 单元测试用例,对代码进行即时检查
- 3) 提供了进行黑盒测试、模型测试和系统测试的快速途径
- 4) 确认并阻止代码中不可捕获的异常、函数错误、内存泄漏、性能问题、安全弱点的问题
- 5) 监视测试的覆盖范围
- 6) 自动执行回归测试
- 7) 支持 DbC 编码规范
- 8) 检验超过 350 个来自 java 专家的开发规范
- 9) 自动纠正违反超过 160 个编码规范的错误
- 10) 允许用户通过图形方式或自动创建方式来自定义编码规范
- 11) 支持大型团队开发中测试设置和测试文件的共享
- 12) 实现和 IBM Websphere Studio /Eclipse IDE 的安全集成

第二章

本章说明了如何安装 jtest full 版本, full 版本包括 jtest 和 eclipse, jtest 被集成在 eclipse 框架内部。

1. 在 windows 安装步骤:

- 1) 双击 jtest 自解压文件 jtest5_win32.exe

- 2) 按提示步骤进行, 选择【complete jtest installation】,一直到结束安装
2. 在 unix 安装步骤:
 - 1) 将安装文件 jtest.linux.tar.gz 或 gzip -dc jtest.solaris.tar.gz 拷贝到即将安装 jtest 的目录
 - 2) 解压档案文件:
linux: gzip -dc jtest.linux.tar.gz | tar xvf -
solaris: gzip -dc jtest.solaris.tar.gz | tar xvf -
解压后, 出现一个叫做 jtest 的目录, 包含了 jtest 安装后的全部文件。
运行 jtest 时, 执行 ./jtest 命令即可。

注册: 获得正版 jtest 安装程序后, 只有正确注册才可以使用, 方法也很简单, 这里不详述。

第三章

本章是 jtest 的快速指南部分, 读者可迅速掌握 jtest 的基本功能及使用方法; 包含的主题如下:

- 1) 第一课: 创建一个实例项目
- 2) 第二课: 检查代码规范
- 3) 第三课: 自动修复代码规范错误
- 4) 第四课: 访问代码规范的描述信息
- 5) 第五课: 忽略/取消忽略报告的错误
- 6) 第六课: 运行 jtest builtin 配置
- 7) 第七课: 以 fly 方式检查特定的编码标准或一组标准
- 8) 第八课: 清除错误信息
- 9) 第九课: 创建、执行、扩展 junit 测试用例
- 10) 第十课: 通过实例配置来修改代码的检查规范
- 11) 第十一课: 创建并运行一个简单的用户自定义 jtest 配置
- 12) 第十二课: 执行回归测试
- 13) 第十三课: 检测内存泄漏

一、 创建一个实例项目

jtest 默认情况下, 可以创建一个 java 实例项目, 本章里, 我们也利用该项目演示。用户在使用自己的代码时, 就不必创建这样的项目来使用 jtest, 直接利用在 eclipse 里创建的 java 项目即可。

目的:

演示如何创建 jtest 实例项目

步骤:

- 1) 选择 file>new>jtest example project
- 2) 默认项目名称是 jtest example, 点击 finish 即可

项目创建完成, 并显示在 jtest 视窗里。Package explore 显示在工作台左侧, 此时只列出一个 jtest example 项目。如何在 java 视窗里查看项目呢?

- 1) 点击工作台左上方快捷条里的 java perspective 按钮; 如果 java perspective 按钮不在, 点击快捷条里的 open perspective 按钮, 选择 other, 然后选择 java, 打开即可。

2) 或者在快捷条里点击 java perspective 按钮进入 jtest 视窗，如下：



——>jtest 视窗



——>java 视窗



——>open 视窗

二、检查代码规范

通过检查代码规范，用户可以避免一些将来可能导致软件功能、性能或安全方面的问题。

目的：

演示如何检查代码规范，并浏览报告的代码规范错误。

步骤：

这里我们检查 simple 这个类（在 jtest example 项目里的 example.eval 包里）是否符合默认的 java 编码规范。

1) 选择 simple.java 源文件



2) 在工具栏里点击 play 按钮，即

默认情况下，点击这个按钮，jtest 会将检查代码规范和单元测试一并执行；后续课程里，我们会说明如何执行特定的测试，这里我们集中讲述代码规范分析和分析结果。

3) 测试运行结束后，在 jtest 运行面板里选择 standards 标签，这里包括如下关于编码规范的内容：

- 代码规范检查耗费的时间
- 被检查的文件个数
- 运行失败的次数
- 发现的错误个数
- 忽略掉的错误个数
- 违反编码的规范个数

4) 关闭运行窗口。

5) 在 jtest 工作台右下方，检查 errors found 视窗。如果该视窗不在，选择 window>show view>other，选择 jtest>errors found，即可。

6) 在 errors found 视窗，打开 example.java 分支，前六个报告的错误是代码规范错误，每个代码规范错误都包含在引起错误的代码行数和简单的错误描述信息。

7) 双击[Line: 54] Text label 'case10' may be typo for 'case 10'节点，simple.java 这个文件的编辑器自动打开，并且违反代码规范的那一行会高亮显示，鼠标也自动定位在错误附近。

三、自动修复代码规范错误

jtest 能自动修复大多数它捕获的代码规范错误，它利用集成的 eclipse quick fix 特性来实现此功能。可以自动修复的代码规范错误，在 jtest configurations 面板的 standards 标签里，都被标记成黄色球形图标。

目的：

演示如何利用 jtest quick fix 特性来自动修复编码规范错误。

步骤：

- 1) 在errors found 视窗里，扩展simple.java节点，双击[Line: 54] Text label 'case10' may be typo for 'case 10'这个错误，编辑器自动打开，错误行高亮显示，并且编辑器的左侧有个黄色球形图标标记这个错误，这个图标的出现表示能够利用 quick fix 选项来自动纠正代码规范错误。
- 2) 在这个黄色球形图标上点击一下，弹出两个选项窗口，一个是针对这个编码错误的quick fix，另一个提供了被违反的编码规范的描述信息。
- 3) 双击fix选项，就会自动纠正编码规范错误，jtest也会重新定位编辑器里的代码。针对这个例子，case10会替换case 10，以后再测试的话，就不会再报错了；而且，jtest也自动删除了在errors found视窗里关于这个错误的信息。
- 4) 保存代码。

四、访问代码规范的描述信息

jtest 能自动检查超过350个编码规范和任何数量的用户自定义规范。每条规范都有一个对应的描述信息，用来帮助用户理解自己的代码为何偏离了正确的规范。

目的：

演示如何深入了解报告的编码规范错误信息

步骤：

- 1) 在errors found视窗，扩展simple.java 分支
 - 2) 右键单击[Line: 53] case 0 is missing either "break", "return", or /*falls through */这个错误，选择view rule description，jtest就会显示在builtin help窗口里显示关于这个规范的描述信息
 - 3) 浏览之后，点击back按钮即可
- 所有的编码规范的描述信息都可以在jtest的帮助里获得。

五、忽略/取消忽略报告的错误

用户在编码时，有些违反编码规范的错误可以忽略不计，那么可以通过此功能来实现，以后再测试时，不会再提示错误信息。如果想针对一些特定编码规范全部忽略，我们推荐最好修改jtest configurations，以便jtest测试时不对其检查。

目的：

演示如何忽略/取消忽略报告的错误。

步骤：

针对simple.java类，想要忽略Utility class does not have a "private" default constructor: 'Simple' 这个错误：

- 1) 在errors found视窗里，扩展simple.java分支
- 2) 右键点击Utility class does not have a "private" default constructor: 'Simple' 这个错误，选择suppress error选项

- 3) 在弹出对话框里输入个短语或句子表示为何要忽略这个错误。针对此例，输入exploring suppressions；关闭对话框，相应的编码规范错误从errors found里清除。将来再测试时，该错误信息会报告在suppressed messages视窗里。

Suppressed messages如果不在，也可通过相同的方法从jtest>show view>suppressed messages打开，这个视窗有如下信息：

- message: 忽略掉的 jtest 错误
- reason: 错误被忽略掉的原因
- resource: 忽略错误的源文件
- user: 执行忽略错误的操作者
- date: 执行忽略错误的日期

如果想取消忽略这个错误：

在suppressed messages视窗里，右键点击**CODSTA.UCDC: Utility class does not have a "private" default constructor: 'Simple'** 这个错误，选择remove suppression即可。

六、运行jtest builtin 配置

jtest configuration是一个定义了用户想要测试的内容设置的集合。每次jtest运行测试，都会用指定的jtest configuration（如果没有选择特定的jtest configuration，会执行默认的jtest configuration）；一般jtest configuration会确定以下一些设置参数：

- 执行的测试类型（编码规范检查、测试用例生成、测试用例执行等）
- 需要检查的编码的规范
- 自动生成测试用例时需要的参数
- 每个测试执行的范围（覆盖多少行等）

jtest包含一系列由jtest开发者预定义的builtin配置，其中有一个java coding conventions（java编码风格）是基于sun公司的java编码风格，以此来检查指定的编码规范，详情参阅<http://java.sun.com/docs/codeconv/html/CodeConvTOC.doc.html>。

目的：

演示如何运行jtest里特定的jtest configurations。

步骤：

这里我们检查simple.java类是否符合sun公司的java编码风格

- 1) 选择simple.java源文件
- 2) 打开play下拉菜单，选择jtest using>builtin>coding standards>code conventions for the java programming language by sun，运行完成后，打开一个对话框
- 3) 关闭对话框
- 4) 在errors found视窗里查看该文件违反sun编码风格的地方

七、以fly方式检查特定的编码标准或一组标准

如果用户想快速检查代码是否符合单一代码规范，或一组规范，可以采用fly方式来检查代码，而不必采用jtest builtin configuration或创建自定义的jtest配置。

目的：

演示如何执行fly方式下的编码规范检查。

步骤:

- 1) 选择simple.java源文件
- 2) 打开play下拉菜单, 选择jtest using>builtin>coding standards rules>formatting>check all rules in this category(也可选择其他的选项)
- 3) 查看errors found视窗里是否出现错误信息

八、清除错误信息

如果用户想要清除掉tasks视窗或errors found视窗里的错误信息, 需要学习本课程。这些清除掉的信息只是临时删除, 下次运行出错, 依然会显示出来。

目的:

演示如何清除掉tasks视窗或errors found视窗里的错误信息。

步骤:

利用shift或ctrl键选择错误信息, 右键点击错误信息, 可选择delete error或删除all/clear all。

九、创建、执行、扩展junit测试用例

jtest 能自动生成并执行junit测试用例, 来发现代码运行的未处理异常, 用户也可以扩展这些用例来增强测试覆盖范围, 检查代码单元级功能、子模块、模块以及系统级功能。

目的:

演示如何自动生成并运行junit测试用例, 如何查看和验证测试结果, 以及如何修复发现的错误。

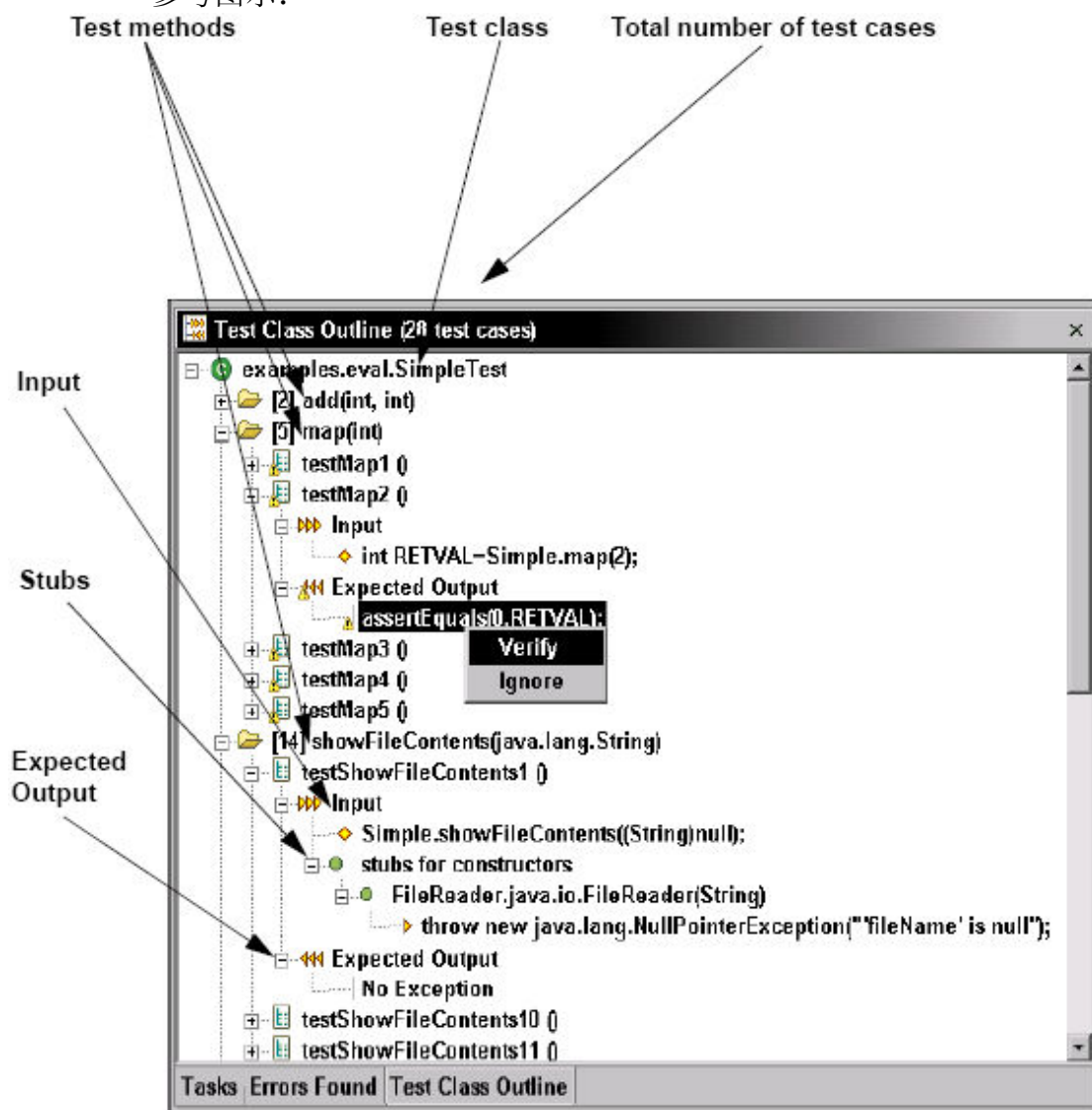
步骤:

- 1) 选择simple.java源文件
- 2) 点击play按钮
- 3) 查看运行后的对话框(这里run标签表示运行了13个用例, error标签发现了一个错误, failures标签里没有运行失败的用例)
- 4) 关闭该运行窗口
- 5) 注意这里jtest把生成的junit测试用例文件simpletest.java加到了一个新项目jtest example.jtest的examples.eval包里
- 6) 在error found视窗里, 右键点击[Line: 107] java .lang. NullPointerException这个错误, 选择quick fix选项
- 7) 按用户意愿自动修复后, 保存代码
- 8) 可通过选择play下拉菜单的Jtest Using> Builtin> Generate and Run Unit Tests, 重新运行测试
- 9) 通过打开simple.java文件, 可以在编辑器里修改测试代码, 以增强测试; 辅助test class outline视窗, 可更好的查看测试代码的结果。如果test class outline视窗不在, 可通过Jtest Perspective>Jtest> Show View> Test Class Outline打开
- 10) 扩展test class outline分支, 能看到每个用例的输入inputs和结果outcomes, 如果知道了每个方法的正确结果, 那么可以:
 - 对每个正确的结果outcome, 可右键点击outcome节点, 选择verify, jtest会将//unverified注释从这个用例文件里除掉, 在以后

测试里，jtest就会检查取该值的输出结果，如果出错则报告错误。

- 对于不正确的outcome，点击outcome节点，会在编辑器里修正，jtest在以后测试里同样检查正确的值，如果出错则报错。
- 对于用户不想让jtest在以后的测试里检查的outcome，可右键点击outcome节点，选择ignore，则将其注释掉，以后测试里将其忽略。

参考图示：



十、通过实例配置来修改代码的检查规范

前述的实例配置默认是全局配置，即点击play时jtest执行的测试；用户可以修改该配置。

目的：

演示如何自定义jtest实例配置增强测试规则

步骤：

- 1) 从菜单中选择jtest-jtest configuration或点击play下拉菜单选择该项，所有可以配置项显示在左侧面板，user defined分支可以修改，builtin项可以查看和拷贝，但不能修改。

- 2) 选择example configuration项，注意此项有个红色G的标志，表示该项被设置成全局配置。
- 3) 打开standards标签，这里是用户可以激活和关闭的代码标准规范
- 4) 点击internationalization复选框，将会提示jtest配置检查并阻止国际化错误以后jtest执行测试时，就会检查国际性错误；如果想取消该检查标准，按照同样的步骤取消复选框即可。

十一、创建并运行一个简单的用户自定义jtest配置

jtest预设置的jtest configurations是基于开发者使用的最普遍的测试场景，如果想自定义测试，可以修改builtin配置或者创建用户定义的jtest配置，我们推荐每次执行不同的测试项目时都设置不同的测试场景。

目的：

演示如何创建一个简单的jtest对类编码的检查规范

步骤：

- 1) 打开jtest-jtest configuration面板
- 2) 选择user defined项
- 3) 点击new，表示要创建一个新的jtest测试场景，jtest添加一个新目录叫做example configuration (1)
- 4) 输入该场景的名字，例如输入metrics，这里我们让jtest只检查类规范
- 5) 打开standards标签
- 6) 点击disable all visible rules按钮，先关闭所有标准；如果看不到这个按钮，扩大整个面板就可以看到了。
- 7) 点击class metrics复选框激活全部的类编码标准；扩展class metrics节点可查看具体规范
- 8) 打开generation标签，清除enable unit test generation复选框；这是为了让这个自定义的测试场景只集中在代码规范上，如果要生成测试用例，运行已经存在的jtest配置即可。
- 9) 打开execution标签清除enable unit test execution复选框，目的和上一个相同
- 10) 点击close，提示是否保存时，选择yes
- 11) 运行该测试场景时，点击要测试的项目，从play下拉菜单里选择user defined-metrics即可

十二、执行回归测试

当jtest第一次运行一个单元测试时，它会创建一个关于当前测试类的功能快照，并以junit格式的测试用例记录了类的行为。实质上，它自动创建了一个回归测试，当一个测试类修改了，可以重新运行测试用例来检查是否出现错误。

目的：

演示jtest如何执行回归测试

步骤：

- 1) 打开simple.java源文件的编辑器
 - 2) 将add()方法里的“+”改成“-”，下一行应该变成return 11-12
 - 3) 保存源文件
 - 4) 选择simple.java源文件
- 点击play按钮运行回归测试；jtest将报告错误junit.framework.AssertionFailedError:

expected:<14> but was:<0>。这个错误提示我们add()方法的功能自从上次测试后改变了；如果该错误不是故意的，例如排版会敲错，我们将要纠正该错误恢复到从前的正确情况；如果是故意这样修改，那就要纠正期望的outcome。

十三、检测内存泄漏

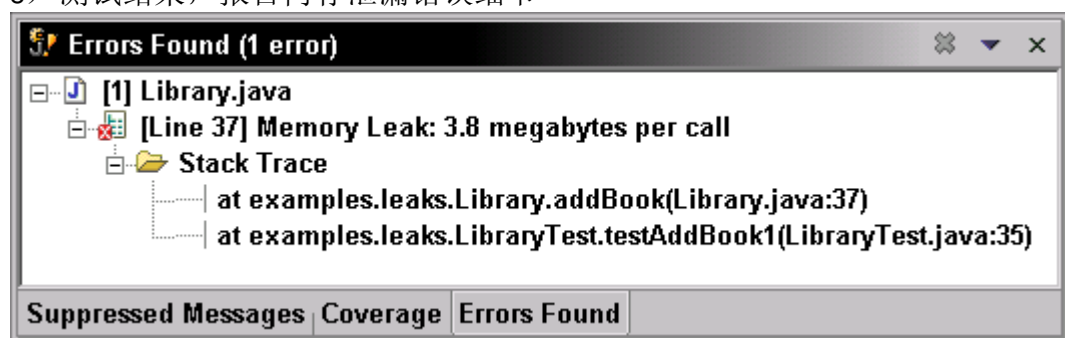
jtest能够在执行单元测试时检测内存泄漏。如果测试结束，内存仍然在使用中，jtest将会报告内存泄漏。

目的：

演示jtest如何在测试执行中检测内存泄漏

步骤：

- 1) 打开example configuration
- 2) 打开execution标签
- 3) 打开options子标签
- 4) 激活detect memory leaks选项
- 5) apply并close
- 6) 选择library.java源文件：打开jtest example项目—example.leaks—library.java
- 7) 从play下拉菜单里选择Jtest Using> User-Defined> Example Configuration
- 8) 测试结束，报告内存泄漏错误细节



- 9) 双击[Line 37] Memory Leak: 3.8 megabytes per call，在源文件里打开代码，找到引起内存泄漏的代码行——每次addbook()方法被同样的参数调用，第37行代码分配的内存没有被释放
- 10) 取消文件末尾的hashCode()和equals()方法的注释，以此来消除内存泄漏错误即可。

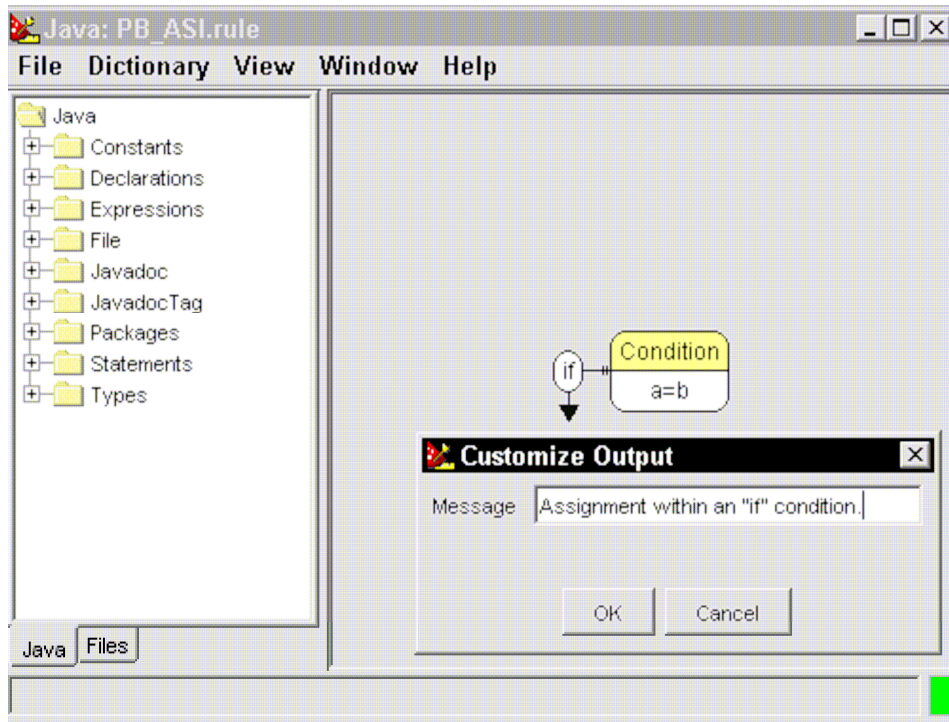
第四章

本章是一些概念，英文书中列举的一些重要概念在前面都有所介绍，另外一些对实际jtest用户用处不大，以后我会写一个如何学习jtest的文档，在那里再仔细阐述。这里有一个概念，前面没有提到，这里简单说一下。

Rulewizard: 是用户自定义jtest代码检查标准的模型，和jtest4.5或早期版本的功能、使用都相同。（关于Jtest4.5版本的使用介绍，本人也有个文档）Jtest能利用Rulewizard自动增加任何有效的代码检查标准，以此，项目团队检查特殊项目的代码需求。

使用Rulewizard，代码规范能通过图形方式（类似流程图）或自动方式（提供代码实例来演示实际的代码规范偏差）来创建。

Rulewizard在jtest architect edition（架构版本）才可以用。这句话是原文说的，但在我们安装的full版里也可以打开，也许是full版里也包括Rulewizard了吧。其界面如下：



如何打开Rulewizard呢？

- 1) 打开**Jtest> Jtest Configurations**
- 2) 打开任何jtest configuration的standards标签
- 3) 右键点击rules树的区域，选择add user-defined rule；Rulewizard界面就打开了，关于它的详细使用（修改、创建、激活等）参考帮助里的view in the Rulewizard UI。不过本人觉得没什么必要，人家parasoft提供的都够多、够全了，我们何必再加这些规范呢？除非真的有特殊需要吧；相反，等你用jtest，就会发现我们开发的代码很多地方都会被jtest认为有错，到时候你可别烦啊，呵呵！

第五章

本章介绍jtest一些任务，主要功能任务都已经在第三章阐述过了；这里只说明一些其他辅助任务。

一、定时执行测试任务：

jtest允许定时执行测试，和其他的自动化工具一样，可以安排夜晚执行测试，第二天来查看测试结果。另外，jtest还可以按照计划时间来创建大批量的测试用例，以不影响白日的工作。

如何定时测试呢？

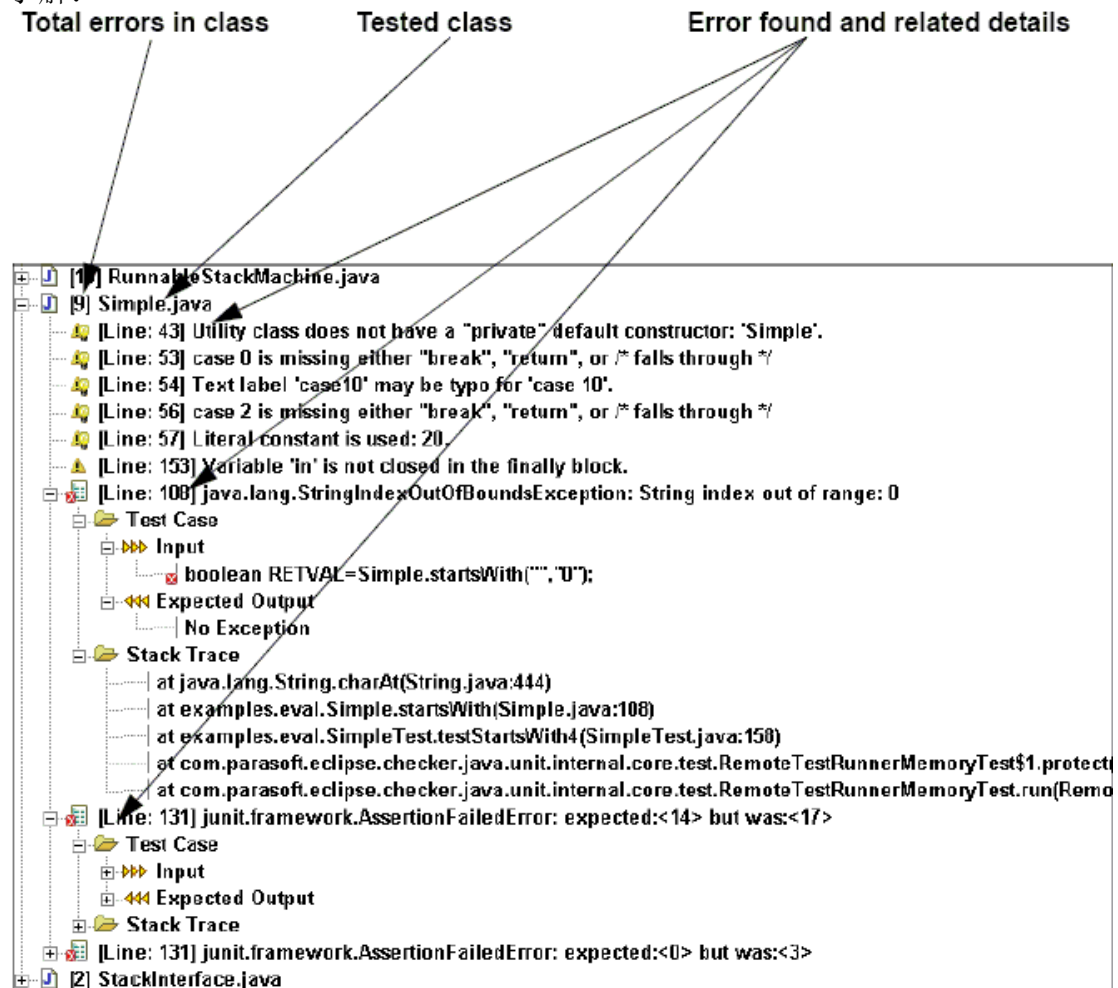
- 1) 选择jtest-preferences
- 2) 选择jtest-scheduled testing
- 3) 激活enable scheduled testing选项
- 4) 在 execution time里输入希望的开始时间：第一个文本框输入小时（0到23），第二个文本框输入分钟（0到59），例如希望在pm 10: 30开始，那么就在第一个框里输入22，在第二个里输入30即可

- 5) 点击configuration to run的edit按钮，选择要执行的jtest configuration
- 6) 下一个edit是选择此执行要覆盖的测试集，即某项目或部分项目文件
- 7) 点击apply

注意：如果jtest主程序没有打开，到了定时时间，测试不会自动运行

二、评估发现的错误：

和第三章介绍的error found一样，这里添加一个参考图示，供读者进一步了解：



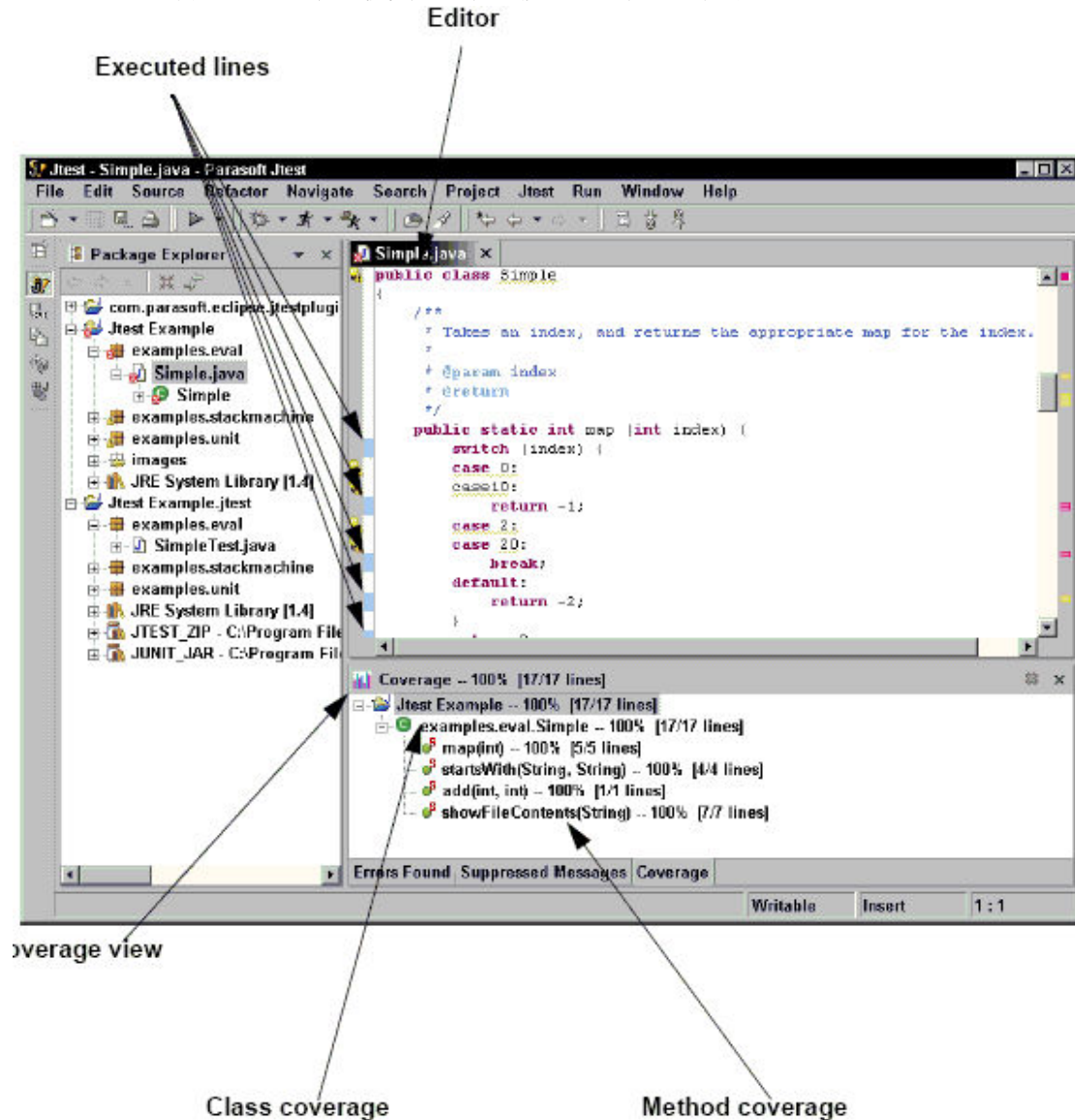
三、查看测试覆盖信息：

jtest 能报告所有 junit 测试用例的覆盖率，因此用户可以评估当前代码被测试的范围，并决定是否需要增加额外的测试用例。一般来说，jtest 能自动创建覆盖了被测试代码 75%的测试用例，有时覆盖率也会达到 100%，有时也会低于 75%。

测试覆盖率的跟踪一般被默认为是激活的。如果想要手工激活：

- 1) 打开 **Jtest> Jtest Configurations**
 - 2) 选择想要修改的测试配置，打开 **execution** 标签，再选择 **option** 子标签，保证 **report code coverage** 选项是激活的
- 测试执行后，想要查看覆盖率信息，有以下两个方法：

- 1) 打开一个测试文件的编辑器，在编辑区域左侧的一个兰色/绿色条表示测试覆盖的行，紫色条表示没有覆盖到的行，没有颜色的行表示不可执行的部分
- 2) 打开 coverage view 面板，显示了每个类和每个方法的覆盖率统计图（覆盖百分比、全部可执行的行、覆盖的行），如图：



总结：关于 jtest 的基本使用，我想上述的内容就够了。白盒测试工具不像 winrunner、robot 等黑盒工具有那么多的预设置或脚本代码编写工作，它的设置简单明了；包括有一些我没有介绍的设置，例如修改测试范围 scope、连接 teamserver 等（本人觉得这些内容也不重要，对使用 jtest 精髓毫无价值），读者只要装了 jtest，按照我介绍的主要功能描述，点击几次自然就融会贯通了。

为了推动我国测试行业的发展，增强测试从业者的职业技能，作为一名普通的测试工程师，我想我们都有责任将自己的经验共享出来，共同交流进步！同时欢迎任何业界朋友对我的作品批评指正，也欢迎更多的人使用本工具（就算学习一下也好），并就任何实际问题开展讨论。

Sincky.zhang 2004-08-20 于北京

欢迎访问我的 blog: <http://blog.51testing.com/index.php?blogId=19>

