XP 单元测试工具 Junit 源代码学习

谢慧强 xiehuiqiang@21cn.com

Package framework

Class & Interface Hierachy

Class Hierarchy

- class java.lang.Object
  - class junit.framework.Assert
    - class junit.framework.TestCase (implements junit.framework.Test)
  - class junit.framework.TestFailure
  - class junit.framework.TestResult
  - class junit.framework.TestSuite (implements junit.framework.Test)
  - class java.lang.Throwable (implements java.io.Serializable)
    - class java.lang.Error
      - class junit.framework.AssertionFailedError

Interface Hierarchy

- interface junit.framework.Protectable
- interface junit.framework.Test
- interface junit.framework.TestListener

Interface Test

主要方法
countTestCases：**统计** TestCases **数目**
run：**运行测试并将结果返回到指定的** TestResult **中**

Class Assert

首先，Assert **提供的** public **方法都可以带或不带自己定义的提示，其次** Assert **中的** Assert **方法是** protected **的，这意外着** Assert **是一个静态类，它提供的方法都是** Static **的。**

public 方法：

assert：保留（deprecated）方法，判断一个条件是否为真
assertTrue：assert 的替代方法，判断一个条件是否为真
assertEquals：用于判断实际值和期望值是否相同（Equals），可以是各种 JAVA 对象。
assertNotNull：判断一个对象是否不为空
assertNull：判断一个对象是否为空
assertSame：判断实际值和期望值是否为同一个对象（==），注意和 assertEquals 区分
fail:直接返回失败，抛出 AssertionFailedError

private 方法：

failNotEquals：主要用于 assertEquals 方法，调用 fail 返回失败提示
failNotSame：主要用于 assertSame 方法，调用 fail 返回失败提示

Class AssertionFailedError

AssertionFailedError 是从 Jdk 提供 Error 类简单继承而来，主要方法如下：

```
public AssertionFailedError (String message) {
    super (message);
}
```

Class Assert 中比较失败都是抛出 AssertionFailedError。

Interface Protectable

这个接口是使用了一种比较少见的用法。
在 Interface 本身只定义了一个方法

```
public abstract void protect() throws Throwable;
```

注意方法 throws 的是所有 Error 和 Exception 的祖先。通过这种定义可以保证运行的时候如果出现任何 Error 和 Exception，都将被抛出而不会导致程序不能继续运行。

Portectable 的接口没有被 framework 包中的任何类实现，它的使用在类 TestResult 中的 run 方法中。以下是 run 方法中代码：

```
protected void run(final TestCase test) {
    startTest(test);
    Protectable p= new Protectable() {
        public void protect() throws Throwable {
            test.runBare();
        }
```

```
        };
        runProtected(test, p);

        endTest(test);
    }
```

这里实际是声明了一个 Anonymous Classes，实现了 Interface Portectable

Interface TestListener

TestListener 的用途和它名称一样，用于监听。主要用于运行时刻监听，BaseRunner(所有运行类，如 TestRunner)实现了这一接口。由于运行是通过 TestResult 来实现，只要调用 TestResult.addListener 就可以增加监听，TestResult 会调用接口中相应的方法，具体见 TestResult。

主要方法：

public

addError：增加错误，注意这里错误应该指测试程序本身的错误或者被测试程序错误，而不是测试失败

addFailure：增加一个测试失败，专用于 AssertionFailedError 的处理

endTest：结束测试

startTest：开始测试

Class TestCase

使用者最主要使用的类，继承 Class Assert,实现 Interface Test。主要方法

public

TestCase：创建本身，可以指定 TestCase 准备运行的测试方法名称，保存在私有属性 fName。

countTestCases：返回 TestCase 数目，直接返回 1

name：deprecated，建议使用 getName,返回 TestCase 当前准备允许的测试方法的名称（私有属性 fName）

run：运行 TestCase,如果没有指定结果存储的 TestResult，将调用 createResu（lt 方法。注意，TestCase 与 TestResult 会有互相调用。整个运行流程如下：

1、TestCase.run 调用 TestResult.run

2、TestResult.run 调用 TestResult .StartTest

3、TestResult.run 创建一个 Anonymous 类，实现接口 Portectable

4、在 Portectable. protect 方法中调用 TestCase .runBare

5、通过运行 Portectable.runBare 调用 runBare，通过 Exception 捕获增加错误及失败报告

runBare：不使用 TestResult 直接运行
runTest：运行测试，注意每调用 runTest 只运行当前 fName 指定的方法
getName：返回 fName
setName：设置 fName

protected

createResult：创建一个 TestResult
setUp：在运行 runTest 前调用
tearDown：在运行 runTest 后调用

Class TestFailure

用于存放测试对比失败信息的类。主要为 Class TestResult 调用。主要属性

```
protected Test fFailedTest;
protected Throwable fThrownException;
```

fFailedTest 存放失败的 TestCase 信息，fThrownException 存放失败提示信息。
主要方法：

public

TestFailure：初始化，对 fFailedTest、fThrownException 赋值。
failedTest：返回 fFailedTest
thrownException：返回 fThrownException
toString：

Class TestResult

TestResult 用于运行并收集测试结果（通过 Exception 捕获），注意 interface TestListener 的所有方法在这里都有同名方法并在同名方法中被调用。

主要属性：

protected Vector fFailures：测试失败报告保存
protected Vector fErrors：测试错误报告保存
protected Vector fListeners：测试监听器保存
protected int fRunTests：运行的测试
private boolean fStop：是否应该停止测试标志，由 stop 方法设置

主要方法

public

TestResult：初试化
addError：synchronized 方法，增加一个错误并向所有监听程序发送错误，调用 TestListener. addError
addFailure：synchronized 方法，增加一个失败并向所有监听程序发送失败，调用 TestListener. addFailure
addListener：synchronized 方法，增加监听程序
removeListener：synchronized 方法，移走监听程序
endTest：结束测试，并通知所有监听程序，调用 TestListener.endTest
errorCount：synchronized 方法，返回错误个数
errors：synchronized 方法，用 Enumeration 返回所有错误
failureCount：synchronized 方法，返回失败个数
failures：synchronized 方法，用 Enumeration 返回所有失败
run：运行测试，创建一个 Anonymous 类，实现接口 Portectable,然后调用 runProtected 方法，可以参看 TestCase 的 run 方法。
runCount：synchronized 方法，返回运行数量
runProtected：实际运行测试
runTests：deprecated 方法，被 runCount 方法替代
shouldStop：synchronized 方法，返回是否应该停止测试。
startTest：开始测试，并通知所有监听程序，调用 TestListener. startTest
stop：synchronized 方法,设置停止标志 fStop 为真。注意是否停止测试 TestReuslt 不负责的，stop 只是简单设置停止标志。
testErrors：deprecated synchronized 方法，被 errorCount 替代
failureCount：deprecated synchronized 方法，被 testFailures 替代
wasSuccessful：synchronized 方法，如果所有运行过的测试方法都通过，返回真，否则为否。

private

cloneListeners：复制 fListeners,主要用于要使用监听列表的 endTest、startTest、addError、addFailure

Class TestSuite

TestSuite 用于将多个 TestCase 集合起来放在一起管理，TestSuite 在增加 TestCase 的时候实际已经将 TestCase 实例化（按包括方法多少做多少次实例化）。

主要属性：

private Vector fTests= new Vector(10)：存放 TestCase 的实例
private String fName：TestSuite 名称

主要方法：

public

TestSuite：初始化，可以选择空、指定名称或包括指定的类。如果是指定的类，那么在 TestSuite 初始话的时候，TestCase 已经实例化并加入到 fTests 中。

addTest：增加一个 TestCase/TestSuit 的实例到 fTests 中。注意由于 TestCase 的实例化实际上只指定一个测试方法,即增加一个 TestCase 的实例是注册了其中一个测试方法，参看 TestCase 类。如参数是一个 TestSuite，则相当于增加了一个子 Suite.

addTestSuite：增加一个子 Suite，实际效果同参数为 TestSuite 的 addTest。

countTestCases：返回 Suite（包括子 Suite）中的 TestCase 实例（测试方法）数量

run：运行测试，注意这里是运行 fTests 中的所有测试，用了 TestResult. shouldStop 方法来判断是否终止运行。实际是调用了 runTest，逐渐

runTest：运行某一 TestCase 或子 Suite 的测试，注意使用了递归。如果参数 test 是一个 TestSuite,会再调用 TestSuite.run

testAt：返回 fTests 指定顺序的 TestCase 或者 TestSuite

testCount：返回 fTests 大小，注意和 countTestCases 的区别

tests：返回 fTests 的内容

setName：设置名称

getName：增加名称

toString：

private

addTestMethod：增加一个测试方法(TestCase 实例)到 fTests

exceptionToString：返回一个 Throwable 中的提示信息

getConstructor：返回指定类的构造函数

isPublicTestMethod：判断一个方法是否是 public 的测试方法，即一个函数是否是 public 的，同时是一个测试方法，测试方法参考下面的 isTestMethod。

isTestMethod：判断一个方法是否是测试方法，既以"test"为前缀、没有参数及返回值。

warning 增加一个错误提示 Testcase 到 fTests 中,注意这里也使用了 Anonymous Class。warning 使用主要考虑的往往在对 TestSuite 进行操作的时候，不会因为有错就终止操作，而是在 run 的时候报告错误

Package extensions

Package extensions 主要包括 TestCase 的各种扩展。

Class Hierarchy

Class Hierarchy

- class java.lang.Object
  - class junit.framework.Assert
    - class junit.framework.TestCase (implements junit.framework.Test)
      - class junit.extensions.ExceptionTestCase
    - class junit.extensions.TestDecorator (implements junit.framework.Test)
      - class junit.extensions.RepeatedTest
      - class junit.extensions.TestSetup
  - class junit.framework.TestSuite (implements junit.framework.Test)
    - class junit.extensions.ActiveTestSuite

Class TestDecorator

TestDecorator 及其子类主要用于在 TestCase 运行前后加入特定操作，进行修饰。不应该使用 TestDecorator 而应该使用 TestDecorator 的子类

主要属性

protected Test fTest：被修饰的 TestCase/TestSuite。

主要方法

Public

TestDecorator：初始化，对 fTest 赋值。

basicRun, run :这两种方法实际都一样，运行测试，为什么要有两个请看 class TestSetup 中的 run 方法代码，run 方法可能被子类覆盖，但 basicRun 不覆盖

countTestCases：返回 TestCases/TestSuite 中测试方法的数量，实际是调用 fTest.countTestCases

getTest：返回 fTest

toString：

Class RepeatedTest

继承 TestDecorator，**重复运行** TestSuite **指定次数。**

主要属性

private int fTimesRepeat：**保存要运行的次数**

主要方法

RepeatedTest：**初始化，对** fTimesRepeat、fTest **赋值**
countTestCases：**返回要允许的测试方法总次数，** TestCases/TestSuite **中测试方法的数量***fTimesRepeat
run：**运行测试**
toString：

Class TestSetup

继承 TestDecorator，**在运行** fTests **前后执行特定操作，注意和** TestCase **中的** setUp/dearDown **不同，** TestSetup **是在** fTests **第一个方法运行前执行** setUp，**在所有方法执行完毕后执行** dearDown,**具体看** run **方法代码。**

Public:

TestSetup：**初始化，对** fTests **赋值**
run：**运行测试，声明了一个** Anonymous Classes，**实现了** Interface Portectable
setUp：**运行前要执行的操作，注意应该和** fTests **定义的类无关**
teardown：**运行前要执行的操作，注意应该和** fTests **定义的类无关**

Class ActiveTestSuite

继承 TestSuite，**在不同线程中运行测试方法，在所有方法运行完毕后在关闭线程。**

主要属性

private volatile int fActiveTestDeathCount：**保存已经运行完毕的线程数量**

主要方法

Public:

run：覆盖父类函数，运行测试。增加对 fActiveTestDeathCount 置 0，和线程结束控制函数

runTest：覆盖父类函数，运行测试方法，主要是先创建一个 Thread，然后运行测试。

waitUntilFinished：判断是否应该结束 Thread，直到 fActiveTestDeathCount 等于 testCount，即 TestSuite. fTests 大小

runFinished：由 runTest 调用，fActiveTestDeathCount 加一

Class ExceptionTestCase

接触 TestCase，拥有测试 TestCase 方法是否抛出特定 Exception。

主要属性

Class fExpected：保存要抛出的 Exception

主要方法

Public

ExceptionTestCase：初始化，指定 fExpected

Protected

runTest：运行测试，主要是增加了 Exception 捕获

Package runner

Package runner 主要用于定义一个基本抽象的 runner，由 runner 派生出来的有 swing、awt 和 text 三种 TestRunner。

Class & Interface Hierachy

Class Hierarchy

- class java.lang.Object
  - class junit.runner.BaseTestRunner (implements junit.framework.TestListener)
  - class java.lang.ClassLoader
    - class junit.runner.TestCaseClassLoader
  - class junit.runner.ClassPathTestCollector (implements junit.runner.TestCollector)
    - class junit.runner.LoadingTestCollector
    - class junit.runner.SimpleTestCollector
  - class junit.runner.ReloadingTestSuiteLoader (implements junit.runner.TestSuiteLoader)
  - class junit.runner.Sorter
  - class junit.runner.StandardTestSuiteLoader (implements junit.runner.TestSuiteLoader)
  - class junit.runner.Version

Interface Hierarchy

- interface junit.runner.FailureDetailView
- interface junit.runner.Sorter.Swapper
- interface junit.runner.TestCollector
- interface junit.runner.TestSuiteLoader

Interface FailureDetailView

显示失败信息接口。主要方法有：
getComponent：**返回用于显示** TraceView **的** Component
showFailure：**显示失败，参数为** TestFailure
clear：**清除** view

Interface TestSuiteLoader

定义如何装入一个 TestSuite，**主要方法**
load：abstract **方法，**throws ClassNotFoundException，**装入** TestSuite
reload：abstract **方法，**throws ClassNotFoundException，**重新装入** TestSuite

Interface TestCollector

用于收集 TestCase/TestSuite **的显示名称。只有一个方法**

collectTests：以 Enumeration 返回所有 TestCase/TestSuite 的显示名称。

Class Sorter 和 interface Sorter. Swapper

Class Sorter 用于提供一个快速排序法，interface Swapper 是一个内部接口，根据注释说明，Sorter 由于与 JDK 1.1.7 兼容性问题，Sorter 不能支持 jdk2 的 collection 类。

interface Sorter. Swapper 只有一个方法

swap：进行替换

Class Sorter 只有一个方法

sortStrings：对字符进行排序

由于 interface Sorter 没有实现类，Sorter 不能直接使用

Class Version

Class Version 用于控制 Junit 版本，主要方法只有一个：

id：返回版本号码

Class StandardTestSuiteLoader

实现 Interface TestSuiteLoader。Class StandardTestSuiteLoader 是 JDK2 本身 ClassLoader 的 TestSuite 装载，同一个类只能装入一次。

主要方法：

load：装载

reload：重新装载

Class TestCaseClassLoader

继承 JDK2 标准类 java.lang.ClassLoader，用于取代系统确省的类装入类。主要是实现只从指定的路径中装入类，对于部分系统 Package 或其他想使用系统装入而不是使用 TestCaseClassLoader 装入的，可以修改 defaultExclusions 或将其写入到文件

excluded.properties 中，系统默认使用系统装入的 Package 有 junit.framework、junit.runner 、 junit. extensions ， 文 件 excluded.properties 必 须 和 Class TestCaseClassLoader 放在一起。在 3.72 版本中的 TestCaseClassLoader 不能装入 jar 文件中类。

主要属性

private Vector fPathItems：类查找路径存放
private String[] defaultExclusions：确省要排除 junit 的三个 package
static final String EXCLUDED_FILE：用户自定义要排除的类
private Vector fExcluded：要排除的路径

主要方法

Public:

TestCaseClassLoader：使用系统默然或指定的类查找路径（使用系统默认路径分割符号的字符串）初始化，在初始化的时候会调用 scanPath 将搜索路径加到 fPathItems 中，调用 readExcludedPackages 函数，将 defaultExclusions 及 EXCLUDED_FILE 中定义的要排除的类加入到 fExcluded 中

getResource ：返回指定资源的 url 路径，调用 JDK2 系统的 ClassLoader.getSystemResource。资源可以是 audio,icon 等，具体可以参看 JDK2 的类说明

getResourceAsStream：以输入流的方式返回指定的资源

isExcluded：判断一个类或包是否应该排除、使用系统标准装入

loadClass：synchronized 方法。装入一个类，覆盖父类方法，请参看 JDK2 中 ClassLoader. loadClass 定义，对于在 fExcluded 的类，采用 findSystemClass 直接装入。其他的将在 fPathItems 定义的路径中寻找（调用 lookupClassData）,找到了就使用 defineClass 将 lookupClassData 返回的字节流转为类并返回，否则就抛出 ClassNotFoundException

package private(default)

isJar：判断一个文件是否用 jar 或 zip 结尾

private

scanPath：由 TestCaseClassLoader 调用将搜索路径加到 fPathItems 中

lookupClassData：查找指定的类，并将 class 内容以字节流的方式返回，调用了 loadJarData，loadFileData。

loadFileData：从文件中装入类的字节流实际读入是调用函数 getClassData

loadJarData：从 zip 文件中（注意，3.72junit 不支持 jar 文件）读入类的字节流

readExcludedPackages：将 defaultExclusions 及 EXCLUDED_FILE 中定义的要排除的类加入到 fExcluded 中

Class ReloadingTestSuiteLoader

实现接口 TestSuiteLoader，使用 TestCaseClassLoader 来装入类。主要方法：
load：装载
reload：重新装载

Class ClassPathTestCollector

实现 TestCollector 接口，分析 Java Class Path，并对 Class Path 中所有的类（JAR、ZIP 文件除外）进行分析，根据指定的规则找出可以运行的测试类。具体规则见 isTestClass 方法

主要属性：

static final int SUFFIX_LENGTH：保持".class"的长度

主要方法

Public

ClassPathTestCollector：初始化函数
collectTests：以 Enumeration 的方式返回所有可以运行的测试类，调用 splitClassPath 将 Class Path 分解为一个 Vector，调用 collectFilesInRoots 获取类

Package Private(Default)

collectFilesInRoots：搜索参数一中的路径，返回所有类。调用 gatherFiles 实际获取类
gatherFiles：获取指定目录下的所有类，是一个递归函数。
splitClassPath：将 Class Path 分解为一个 Vector

Protected

isTestClass：判断一个类是不是测试类，规则文件名中扩展名为 class,不包括"$"，包含了"Test"。
classNameFromFile：从文件名中获取类名称

Class SimpleTestCollector

继承了 ClassPathTestCollector，从代码来看，实际没有对 ClassPathTestCollector 做任何修改，虽然代码包括 isTestClass，但其中的代码和 ClassPathTestCollector 中是一样的。

Class LoadingTestCollector

继承了 ClassPathTestCollector，修改了判断测试类的方法。

主要属性：

TestCaseClassLoader fLoader：类装入类

主要方法

public：

LoadingTestCollector：初始话，实例化 fLoader

protected：

isTestClass：覆盖父类中的方法，注意还有一个 Package private(Default)的 isTestClass。使用 classFromFile 从文件生成一个 Class 后调用 Package private(Default)的 isTestClass 判断是否是一个测试类。

package private(Default)：

classFromFile：使用 TestCaseClassLoader 从一个文件中装入类

isTestClass：判断一个 Class 是否是一个测试类，判断的依据是是否包括一个在 BaseTestRunner. SUITE_METHODNAME 定义名称的方法（suite）方法或者是具有以下特性：

1、实现了 Interface test，

2、有一个 Public 的 Constructor 方法

3、Modifier 是 public 的

hasSuiteMethod :判断十分有 BaseTestRunner. SUITE_METHODNAME 定义名称的方法

hasPublicConstructor：判断是否有有一个 Public 的 Constructor 方法

Class BaseTestRunner

BaseTestRunner 是一个抽象类，是所有 Runner 的基类。BaseTestRunner 使用类静态代码，在被初始化的时候运行。

主要属性

public static final String SUITE_METHODNAME= "suite"：suite 方法定义
static Properties fPreferences：保存参数
static int fgMaxMessageLength= 500：定义最长 Message 长度
static boolean fgFilterStack= true：是否过虑错误信息
boolfPreferencesean fLoading= true：标志是否使用自定义的 ClassLoad 类

主要方法

Public:

getTest：根据 suiteClassName 名称返回一个 TEST 类，如果类有 SUITE_METHODNAME 定义的方法，调用 suite 方法，否则返回 new TestSuite(testClass)，创建一个新的 suite。
elapsedTimeAsString：将时间转为指定格式的字符串
setLoading：设置 fLoading
extractClassName：从一个字符串中返回 ClassName,主要是处理 VA/Java style
truncate：static 方法，将一个字符串截为指定长度
getLoader：判断并返回使用 ReloadingTestSuiteLoader 还是 StandardTestSuiteLoader，调用 useReloadingTestSuiteLoader 方法
getPreference：从 fPreferences 中返回指定的参数值
inVAJava：判断是否在 VisualAge 环境中
getFilteredTrace：将参数 Throwable t 中包括的信息进行过滤（Filter）并返回处理后的字符信息
filterStack：对字符进行过滤
filterLine：判断字符是否应该过滤

protected：

processArguments：处理传入的参数，如果参数中包括了要运行 TestSuite 则返回 TestSuite 名称，如果参数中包括“-noloading”，则调用 setLoading（false）不使用 ReloadingTestSuiteLoader，使用 StandardTestSuiteLoader。如果参数中包括”-nofilterstack”，设置 fgFilterStack=false，可以使用“-c xxx”指定要测试的类
runFailed：abstract 方法，运行失败时处理
loadSuiteClass：从一个 TestSuite 的名称装入一个类，调用 getLoader 来判断使用 ReloadingTestSuiteLoader 还是 StandardTestSuiteLoader

clearStatus：清楚状态信息，需要子类覆盖
useReloadingTestSuiteLoader：判断是否使用 ReloadingTestSuiteLoader

private：

getPreferencesFile：读取参数配置文件，配置文件名称为 junit.properties，应该放在 System.getProperty("user.home")指定的目录下
readPreferences：将参数配置文件中的内容放到 fPreferences 中

程序中的静态代码完成 fPreferences 的初始话及读取参数文件、设置 fgMaxMessageLength。

Package textui

Package textui 仅有一个类 TestRunner，用于实现文本方式的运行。

Class TestRunner

继承 BaseTestRunner， TestRunner 用于完成文本方式的测试运行。

主要属性：

PrintStream fWriter= System.out：输出流
int fColumn= 0：仅用于 startTest 方法，其实可以不用

主要方法：

Public:

TestRunner：构造函数，可以自己指定输出流
getLoader ：覆盖父类方法，只使用默认的类装入（StandardTestSuiteLoader）
addError：synchronized 方法，增加一个错误
addFailure：synchronized 方法，增加一个失败
doRun：运行指定的 TestSuite 并返回 TestResult
startTest：开始测试
endTest：结束测试
main：运行函数，用于外部直接调用。将参数直接转给 start 方法。主要

参数为

-wait：**设定等待**

-v：**显示版本号**

-c **类名称：要运行的测试类**

具体看 start **方法。**

print：synchronized **方法，输出结果，调用** printErrors(result);

printFailures(result);

printHeader(result);

printErrors：**输出错误**

printFailures：**输出失败**

printHeader：**输出总计**

run：**运行指定的** TestCase/TestSuite

runAndWait：**运行指定的** TestSuite **并等待，参看** pause **方法**

Protected

createTestResult：**创建一个** TestResult

pause：**判断是否要** pause

start：**运行测试，由** main **方法调用。**

runFailed：**运行失败处理方法**

writer：**返回输出流** fWriter