文件编号：NCI/SP-SE05-G01

NCI-单元测试指南

撰 写 人：张仁信

编写时间：2008-7-5

部门名称：研究中心

审 核 人：EPG

审核时间：2008-8-28

杭州新世纪信息技术股份有限公司
HANGZHOU NEW CENTURY INFORMATION TECHNOLOGY CO., LTD.

修 订 记 录

*A – 增加 M –修改 D –删除

版本	日期	图或表或章节号	A M D	标题或简要描述	变更申请号	修订者	批准者
V1.0	7 月 5 日		A			张仁信	李云水

目　　录

前言

我们将分三个部分讲解单元测试内容：

思想篇：我们将从几个方面对单元测试的讲解，使我们从思想上对单元测试的重要，或者开始意识到单元测试的必要性。

实践篇：通过单元测试思想篇的熏陶，使我们开始想为单元测试做点事情了，来吧，让我们开始进入单元测试的世界。

总结篇：你已经开始对单元测试从思想上有了解，也对单元测试做了亲密的接触，这时，需要我们对单元测试进行总结，能让我们更好地做好的单元测试工作。

思想篇

当我们要对编码进行单元测试时，除了事先的单元测试计划之外，我们是否还有考虑一些问题，而这些问题让我们对单元测试有一个全面和深刻的认识，使我们对单元测试有一个更加好的定位。

1. 什么是单元测试

单元测试是研发者编写的一小段代码，用于检验被测代码的一个很小的、很明确的功能是否正确。通常而言，一个单元测试是用于判断某个特定条件（或场景）下某个特定函数的行为。

2. 单元测试做什么

单元测试过程要做什么事，我们从下面几个方面可以得到一些答案：

1) 他的行为和我期望的一致吗？
2) 他的行为一直和我期望的一致吗？
3) 我能够依赖单元测试吗？
4) 单元测试说明我的意图了吗？

3. 为什么要单元测试

为什么要单元测试，它能给我们带来什么直接的好处：

1) 它是一种验证行为。测试来验证程式中的每一项功能都是正确的。
2) 它是一种设计行为。编写单元测试将使我们从调用者观察、思考。
3) 它是一种编写文档的行为。单元测试是一种无价的文档，他是展示函数或类如何使用的最好文档。

4. 不写单元测试的借口

我们有了为什么要单元测试的原因，还要举例一些不写单元测试的借口，如果你也是不写单元测试的，看是否包含在下面的借口中，如果是，是否需要在重新考虑一下，它真的是你不写单元测试的理由吗？

1) 编写单元测试太花时间了。
2) 运行测试的时间太长了。
3) 测试代码并不是我的工作。
4) 我并不清楚代码的行为，所以也就无从测试。
5) 但是这些代码都能够编译通过。
6) 公司请我来是为了写代码，而不是写测试。
7) 假如我让测试员或 QA（Quality Assurance）人员没有工作，那么我会觉得很内疚。
8) 我的公司并不会让我在真实系统中运行单元测试。
9) 我不知道怎么写单元测试。
10) 业务逻辑简单，不值得编写单元测试；
11) 项目没有要求，所以不编写单元测试；

5. 单元测试的测试对象

哪些内容可以纳入为单元测试的对象，这个问题是要在单元测试之前，有一个明确的定义。

测试对象原则：最大的粒度应该控制在一个类级别上，最合适的粒度是控制在一个方法级别上。

对于单元测试的测试对象的范围，我们做了如下汇总：

1) 类（JAVA 范畴）
2) 方法（JAVA 范畴）
3) 功能模型（JFW 框架范畴）
4) 代码模型（JFW 框架范畴）
5) JSP 页面模版（JFW 框架范畴）
6) JAVASCRIPT 函数脚本
7) 存储过程（ORACLE 范畴）

关于何如对上面的测试对象做单元测试，我们将在实践篇进行详细的说明。

6. 怎么编写单元测试用例

在我们进行单元测试之前，要做一些准备工作，那就是编写单元测试用例。单元测试用例是对单元测试的指导，在单元测试用例的参照下，完成单元测试工作。

测试用例是为某个特殊目标而编制的一组测试输入、执行程序以及预期结果，以便测试某个程序路径或核实是否满足某个特定需求。所以，下面将从三个方面进行讲述怎么编写单

元测试用例。

6.1. 输入数据

我们使用一定的规则选择有代表性的数据作为输入。一般输入数据可分为三大类：正常输入，边界输入，非法输入，每大类还可再分为若干小类，划分小类的依据是：同一小类中每个数据都具有等价的测试效果，这就是平常说的“等价类法”。

正常输入

例如字符串的 Trim 函数，功能是将字符串前后的空格去除，那么正常的输入可以有四类：

前面有空格；

后面有空格；

前后均有空格；

前后均无空格。

边界输入

上例中空字符串可以看作是边界输入。

再如一个表示年龄的参数，它的有效范围是 0-100，那么边界输入有两个：0 和 100。

非正常输入

垃圾数据或使代码不能完成正常功能的数据，如一个文件操作的函数，非正常输入有这么几类：

文件不存在；

目录不存在；

文件正在被其他程序打开。

6.2. 执行程序

执行程序主要是从白盒测试角度对代码进行覆盖测试，一般要达到以下目标为至：

- ✓ 测试到每一个最小语句的代码；
- ✓ 测试到所有的输出结果。

为了实现代码的覆盖测试，我们可以采用一些白盒测试方法进行辅助。

6.3. 预期结果

预期输出

一个完整的测试用例应该有预期输出，预期输出就是程序运行后的预期结果，通常表现在对某些数据的修改，即预期输出要自动判断程序所改写的数据的结果值是否符合预期。程序可能修改的数据包括：

A、返回值；

B、输出参数；

C、成员变量，只考虑函数所改写的成员变量；

D、全局变量，只考虑函数所改写的全局变量；

E、其他数据，如函数改写文件或数据库中的数据，也是一种输出，不过通常难于自动判断是否符合预期，可用人工查看来代替。

7. 有什么好的单元测试方法

SQL 脚本驱动测试方法；
JAVA 语句驱动测试方法；
JS 语句驱动测试方法；
页面事件驱动测试方法。

8. 单元测试模型

在我们的思想篇最后一章节，将要讲述一下单元测试过程。在软件工程中，软件测试过程是一种抽象的模型，用于定义软件测试的流程和方法。单元测试过程是软件测试过程中的一种。在软件工程中，并没有单独对单元测试过程进行描述，但我们可以从软件测试过程中找到适合单元测试的过程。在这里我们要讲述 H 模型。

H 模型将测试活动完全独立出来，形成了一个完全独立的流程，将测试准备活动和测试执行活动清晰地体现出来，如图所示。

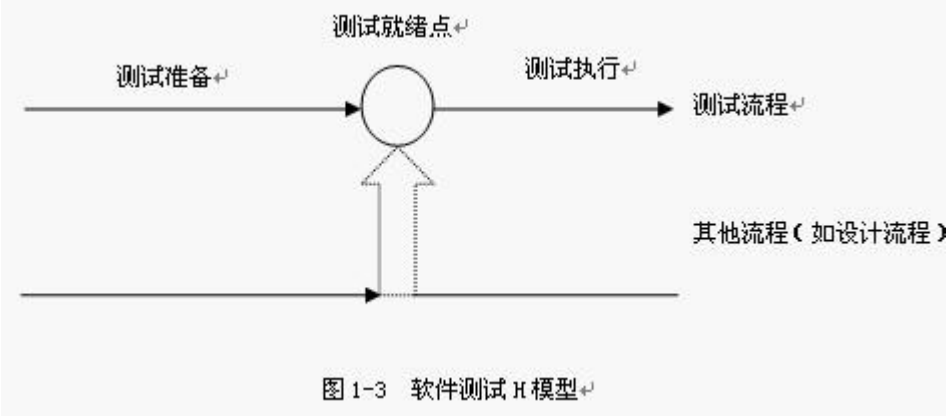


图 1 软件测试 H 模型

这个示意图仅仅演示了在整个生产周期中某个层次上的一次测试“微循环”。图中标注的其他流程可以是任意的开发流程。例如，设计流程或编码流程。也就是说，只要测试条件成熟了，测试准备活动完成了，测试执行活动就可以（或者说需要）进行了。

实践篇

在单元测试思想篇的熏陶下，我们已经对单元测试有了一个比较全面的认识，也应该具备了单元测试的必要条件。接下来，我们就开始单元测试的实践篇。

9. 准备阶段

在开始单元测试之间，我们需要做三个方面的准备：

- 单元测试用例
- 单元测试脚本
- 单元测试数据库

10. 类与方法的单元测试（JAVA 范畴）

类与方法的单元测试，一般使用 JAVA 语句驱动测试方法：

- ✓ 在另一个类中对类或方法进行调用测试；
- ✓ 在自身类的 main()函数调用测试类或方法。

在另一个类中对类或方法进行调用测试

例如：

我们编写了一个技术对象服务工厂类（TOServiceFactory），内容如下：

```
/**
 * 技术对象服务工厂类 TOServiceFactory
 * <p></p>
 * 描述：定义技术对象服务工厂函数<br>
 * 版权：Copyright (c) 2008 杭州新世纪信息技术股份有限公司
 * <p></p>
 * @author zhangrx
 */
public class TOServiceFactory {

	/**
	 * 说明：获取技术对象服务
	 * @return
	 */
	public static TOService getTOService() {
		return (TOService)JFWManager.getBean("toservice");
	}

}
```

在这个类中包含了方法 getTOService()获取技术对象服务。现在，我们通过另一个实现类的调用来测试这个方法正确性。

调用实现类如下：

```
package com.nci.epms.equipment.business.adu;

/**
 * @company 杭州新世纪信息技术有限公司
```

```
 * @author  龚健健
 * @date 2008-4-29
 * @version 1.0  说明：创建单设备调整的功能类
 *
 */
public class SingleEquipment extends MasterDetail {
      ……
      public void executeAfter(JFWContext context, String moduleName,
                  DataSetOp dataSetOp, String actionFlag) throws OperationException {
            if ("ADD_SAVE".equalsIgnoreCase(actionFlag)) {
                  ……
            } else if ("UPDATE_SAVE".equalsIgnoreCase(actionFlag)) {
                  TOService toService = TOServiceFactory.getTOService();
                  System.out.println("toService content:"+ toService);
                  try {
                        ……
                  } catch (Exception ex) {
                        ex.printStackTrace();
                  }
                  context.put("JFW_FORWARD", "update");
            } else if ("ADD".equalsIgnoreCase(actionFlag)) {
                  ……
            }
      }
      ……
}
```

在另一个类中使用系统输出语句判断是否获取正常数据。

在自身类的 main()函数调用测试类或方法

例如：

我们在工具类中编写了一个字符串的 Trim 函数，功能是将字符串前后的空格去除；内容如下：

```
package com.nci.epms.equipment.util;

public class StringParser {
      private String string;
      private DoInString doIn;
      private StringBuffer buffer;
      private StringBuffer result;
      public StringParser(String str,DoInString doIn) {
            this.string = str;
            this.doIn = doIn;
      }

      /**
```

```
     * 说明：将字符串前后的空格去除
     * @param str
     * @return
     */
    public static String trim(String str) {
        String tmp = null;
        if(str!=null && str.length()>0) {
            //除去字符串前后的空格
            str = str.trim();
        }
        return tmp;
    }

    /**
     * 主方法
     * @param args
     */
    public static void main(String[] args) {
        //单元测试开始
        String testValue = null;
        //正常输入
        testValue = " AAA ";       //前后有空格
        System.out.println("测试结果为："+StringParser.trim(testValue));
        testValue = "AAA ";        //后面有空格
        System.out.println("测试结果为："+StringParser.trim(testValue));
        testValue = " AAA";        //前面有空格
        System.out.println("测试结果为："+StringParser.trim(testValue));
        testValue = "AAA";         //前后无空格
        System.out.println("测试结果为："+StringParser.trim(testValue));
        //边界输入
        testValue = " ";           //空内容
        System.out.println("测试结果为："+StringParser.trim(testValue));
        //非法输入
        testValue = null;          //空值
        System.out.println("测试结果为："+StringParser.trim(testValue));
        //单元测试开始
    }
}
```

在 main()方法中对 trim 函数做了单元测试。

11. 功能模型、页面模型与 JSP 页面模版的单元测试

功能模型、页面模型与 JSP 页面模版的单元测试，将通过框架的层面进行解决，这里将不在论述了。

12. JAVASCRIPT 函数脚本的单元测试

JavaScript 测试主要分为返回值测试和执行过程测试。

JavaScript 代码运行在客户端浏览器上，是一种解释型的脚本语言，测试调试有一定的局限性，不像 JAVA，C++等传统编程语言有强大的编辑工具支持，可以对代码进行运行期联调，实时跟踪代码运行情况。一般情况下无法对 JavaScript 进行运行期联调，主要通过在相关代码行中手工写入打印、提示调试信息来辅助判断程序运行情况。

- 函数返回值测试

如果函数有返回值，就可以通过“提示返回值”的方式来判断程序运行的情况。例如：

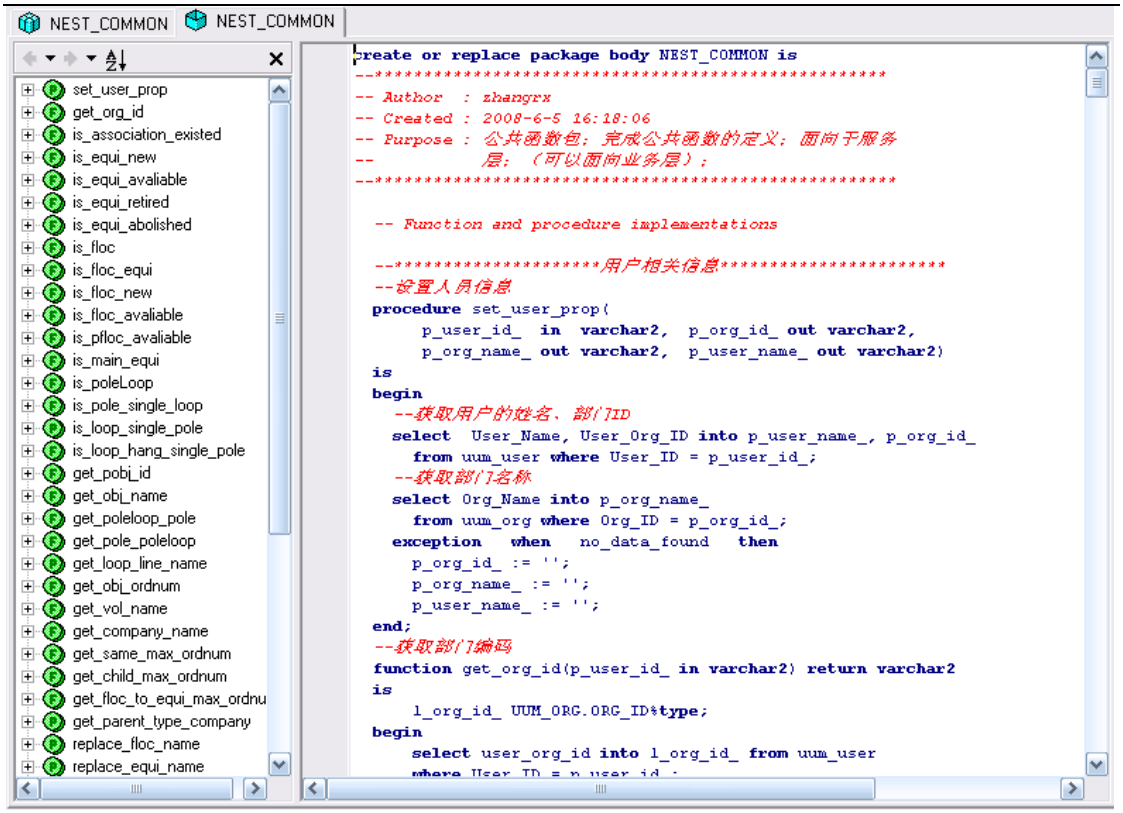
```
function check(v){
    return v === null || v === undefined || v === '' || v === 'null';
}
alert(check('')); //测试返回值
```

- 代码执行过程测试

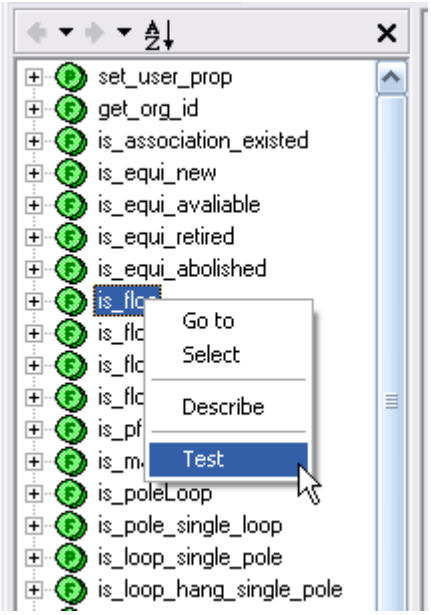
如果要判断函数执行过程是否按照预期的设计在运行，就可以通过“提示状态信息”来判断，首先在函数执行路径的关键点上写入 alert（）提示语句，尽可能的将能反应当前执行情况的状态信息提示出来，然后调用函数，观察提示信息，判断函数执行情况。

13. 存储过程的单元测试（ORACLE 范畴）

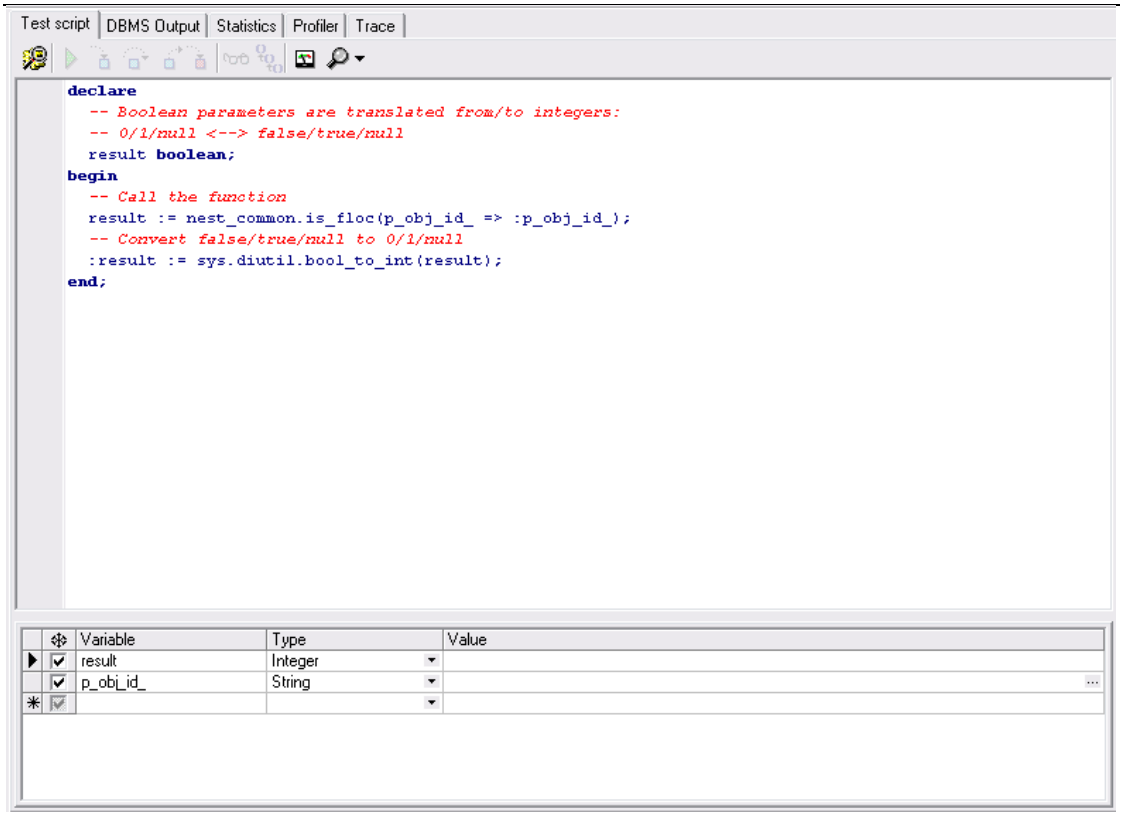
关于存储过程的单元测试，我们可以借助第三方工具来协助完成存储过程的单元测试工作。在这里，我们将使用 PLSQL Developer 工具来完成存储过程的单元测试。



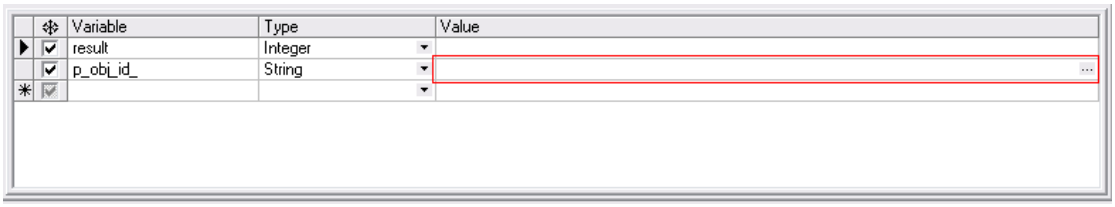
上面就是通过 PLSQL Developer 工具查看的存储过程包图。当我们需要对某个过程/函数进行单元测试时，只需要对所要测试的过程/函数点击右键，如下图：



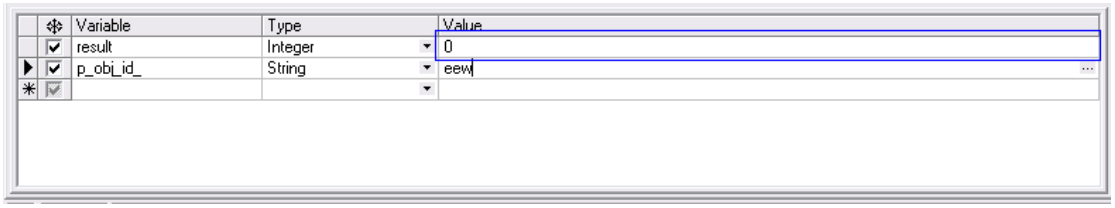
点击【Test】菜单，进入了过程/函数的测试调试页面，页面如下：



在测试调试页面，分成了两个部分，上面为过程/函数的调试语句，下面为调试输入窗口，我们将在输入窗口的红框内输入数据（单元测试用例中的测试输入数据），如下图：



根据不同的输入测试数据，过程/函数将产生不同的结果，结果将出现在蓝框中，如下图：



这样便完成了存储过程的单元测试工作。

总结篇

14. 总结

总结篇是最后一篇。在我们经过了之前的思想篇，实践篇，已经使我们对单元测试从理论和实践上都存在了一定的认识。在总结篇，我想大家可否在提升一个阶段。我们已经知道单元测试的理论了，也进行实践操作，那怎么做可以更好地完成单元测试工作，下面几点，可以让我们有好的思考：

1）组织结构应该保证测试组参与单元测试；

2）加强单元测试流程规范性；

3）制订单元测试的过程定义；

4）单元测试工作产品必须纳入配置管理；

5）必须制订覆盖率指标和质量目标来指导和验收单元测试；

6）加强详细设计文档评审；

7）加强对单元测试人员的技能培训；

8）必须引入工具进行辅助；

9）单元测试者加强对被测软件的全面了解。

单元测试是软件开发过程中非常重要的质量保证手段，加强单元测试对提高软件质量具有非常重要的意义。所以需要我们在思想上重视单元测试。

最后，我们再给一个数据：根据业界的统计，一个 BUG 在单元测试阶段发现花费是 1 的话，到集成测试就变为 10，到系统测试就高达 100，到实际推向市场量产后就高达 1000。