PL_SQL构建代码分析工具之从测试开始 PDF转换可能丢失图片或格式，建议阅读原文 https://www.100test.com/kao_ti2020/462/2021_2022_PL_SQL_E6_9E_84_E5_c102_462383.htm Feuerstein 的“ 构建代码分析工具 ”系列的第 2 部分 在本系列的第一篇文章中，我决定构建一个可以对代码进行质量检查的实用工具：特别是识别 PL/SQL 程序包中具有歧义或者潜在歧义的超载问题。 此外，我还识别数据源（ALL_ARGUMENTS 数据字典视图）和代码（DBMS_DESCRIBE 程序包），以帮助构建实用工具。下一步要做什么呢？ 坦率地说，我自然倾向于打开我所喜爱的集成开发环境 （IDE） 并开始动指如飞地编写代码，投入激动人心的创作之中。我希望在工作时思考并得出结论，不断地应对挑战，使工作成果运行起来，然后对其进行微调。 这种方法有积极的方面（您当然不会有过度设计的风险），但也有很多缺点。首先，如果我所使用的构建方法是直通式构造非线性系统 常被亲切地称为 HRCLS 或 Hercules（一种非常轻型的方法论），我最终不会仅是微调代码。绝对不会，在我将范围缩小到最终目的、目标和实质内容时，我最终将会以重写整个程序一次、两次或三次。但是，尽管可以激动地看到，经过自己的努力，实用工具成形、进步并改观，但这会浪费很多的时间。 对于 Codecheck，我会忍住最初的诱惑。我不会在项目开始的头 60 秒内就编写代码，而是做一个简单的声明：我保证使我编写的代码适用于一个全面的测试计划。 这与避免 Hercules 综合症有什么必然的关系呢？让我们来考虑这种保证的含义： 我将制订一个测试计划，它非常全面，精心策划了数十个（甚至可能有 100 个之多）测试方案，

包括参数的数量、各种数据类型的结合以及缺省值的有无。我要设计、构建和运行测试，这些测试可确定我的代码是否满足测试计划的要求：换言之，设计、构建和运行涵盖我的全部测试方案的测试。 天哪，这听起来确实很有道理，不是吗？我的意思是，谁会一直做所有的这些事情？况且，我们都知道这样的事实：我们之中很少有人实际上会花时间，或在手边有必备的工具去做全面的测试。实际上我认为，世界上至少百分之九十的软件开发人员和开发团队（包括我自己）远远不会完全按上述保证去做。 对我来说，在开发初始时我表明要执着地进行测试，我发现这一点促使我实现了Codecheck. 定义范围，做出假设 在开发一项测试策略并构造测试之前，我需要确定项目的使用范围。用户群通常确定（并经常更改）项目的使用范围。对于 Codecheck 以及我制作的很多其他实用工具，您，开发人员，是我的用户，但我仍然需要确定使用范围，把您的需求作为我的指导。我的目标是产生一个代码体，能立即帮助开发人员提高代码质量，但也可以作为一项稳定而且可扩展的功能，开发人员可以将其加入代码中满足自己的需要。它必须能够处理现实世界中足够的复杂性，这样才有用处，但又不可太复杂，以至于将 Codecheck 开发变成一种全职工作。 当我深入查看 ALL_ARGUMENTS 的内容时，我吃惊地发现那些参数列表会变得非常复杂。例如，在 Oracle9i 的环境中，一个参数可以将记录的结合数组（以前称为表的索引）作为其类型，其中记录的一个字段是另一个结合数组，以此类推。非常坦率地说，我实际上不希望必须编写代码来处理这些参数全部的可能情况。 与此相反，Codecheck 将只基于“ 0级 ”参数来检查

和进行分析：这是指在 ALL_ARGUMENTS 视图中的那些项目，当其出现在程序头时，它们对应于参数列表。实际上，这可能是您为分析超载而所需的全部要素，但是全部的详细资料可能在进行某种其他的代码检查时派上用场。（在撰写此文章时，我已确定了一种需要更多信息的情况：如果您将一个参数定义为表单 %ROWTYPE 的记录，则 ALL_ARGUMENTS 和 DBMS_DESCRIBE 都不会提供其类型的信息。您可能必须比较那些行类型（1 级或更多级）的各个字段的数据类型以识别岐义。天哪。 定义测试策略 开发 Codecheck 的下一步是探究和定义测试策略。我曾提到过，我承诺编写能够满足测试计划的代码，这种承诺影响着开发过程。例如，我在编写如 Codecheck 等实用工具时的第一个冲动是，制作一个程序，接受一个程序包的名称并在屏幕上显示代码检查的结果。为了确定实用工具是否正确工作，我需要查看屏幕上的检查结果并对其进行分析。这听起来如何？熟悉吗？可伸缩吗？通常这种方法甚至对非常简单的程序都无效。 Codecheck 测试计划无疑会有很多测试方案，而其结果并不明确。换言之，除非我记住成功对所有这些测试方案意味着什么，否则我必须依靠手动的、直观的验证（先看这个窗口；然后将其与那个窗口的内容进行比较）。我需要用很长的时间完成一项测试（这是构成责任性的一项相当重要的要素），所以我很少进行测试 如果曾做过测试的话。这与我的保证相违背。 我需要找到一种更快捷方便的测试方法。使用 Java 的许多开发人员转向 Junit.PL/SQL 开发人员则利用 utPLSQL，一种用于 PL/SQL 开发人员的开放源单元测试框架。（申明：我是 utPLSQL 的创建者，尽管其他人现在也帮

助进行实施并制作文档。） utPLSQL 简介 utPLSQL 是一种用于 PL/SQL 程序的测试框架（代码以及使用这种代码的进程的集合）。使用 utPLSQL，您可以构造包含单元测试过程的单元测试程序包，并依照 utPLSQL 的命名规范和测试机制来设计此程序包。然后您只须简单地操作 utPLSQL 来测试程序或程序包。它运行所有测试并自动检测该测试是否成功或失败。它准确指出失败的测试方案，使您更快地确定正确的测试方案并识别应用程序中错误的原因。 这种框架是基于极限编程的单元测试概念 （ www.xprogramming.com ） 以及 Junit （Java 单元测试框架）。下面是一些该方法进行单元测试的基本原则： 在编写代码前编写单元测试。 少做编写和更改，多做测试。 自动执行测试和生成报告：红绿灯式的方法。 下面对行程中的各站作一解释： 调用 utPLSQL 测试过程执行测试程序包。utPLSQL 按照动态 PL/SQL 和命名规范来运行任何设置代码，定位并执行单元测试程序，并进行必要的清除（“ 拆卸 ”步骤）。 单元测试过程调用“ 判断 API ”，它将测试结果与“ 控制 ”条件进行比较。 判断程序将结果（通过或失败）写到下面的结果表中。 tPLSQL 测试读出结果表中的内容，确定该测试的状态。 utPLSQL 根据结果作出报告，或者通过 DBMS_OUTPUT 送到屏幕，或者通过 UTL_FILE 送到一个文件。 让我们来看一个很简单的示例，从而大致了解单元测试程序包的内容。假设我已经创建了一个关于 SUBSTR 的封装，允许在开始和结束点之间请求一个子串（一个简单的函数）。 即使是简单的或看似不重要的程序（如 betwnStr）也需要测试 而实际上我必须考虑大量的测试方案（包括 NULL 开始值、NULL 结束值以及开始大于结束等）。 那些

在测试领域工作的人都知道，测试一个应用程序所需要编写的代码数量常常比应用程序本身更多。对于这个特定的程序包和测试代码的 utPLSQL 风格，您在某些情况下会生成全部的测试代码。实际上，ut_betwnstr 程序包通过对 utGen.testpkg_from_string 过程的调用而生成。 即使您不生成测试程序包，也经常可以找到其他方法，利用最少的代码运行许多基于 utPLSQL 的测试 这正是我用 Codecheck 所做的工作。 将 utPLSQL 应用于 Codecheck 要利用 utPLSQL，我需要构建一个单元测试程序包并调用 utAssert 程序，以确定我的代码是否通过其测试。但是，在进行这些操作之前，我需要精心制作我的测试方案。请记住：测试方案第一，其次才是代码。 现在应该寻找灵感，暂停前进，考虑一下我所提供的实用工具。我希望它能够验证什么？那些能编译但包含歧义超载的程序包的特例是什么？我能检测到什么情况？有效超载的示例是什么？毕竟我需要测试正面和负面的因素。经过一段时间以后，我给出以下的内容： 