目录

软件测试常识

作者：张华

软件开发和使用的历史已经留给了我们很多由于软件缺陷而导致的巨大财力、物力损失的经验教训。这些经验教训迫使我们这些测试工程师们必须采取强有力的检测措施来检测未发现的隐藏的软件缺陷。

生产软件的最终目的是为了满足客户需求，我们以客户需求作为评判软件质量的标准，认为软件缺陷（ Software Bug ）的具体含义包括下面几个因素：

- 软件未达到客户需求的功能和性能；
- 软件超出客户需求的范围；
- 软件出现客户需求不能容忍的错误；
- 软件的使用未能符合客户的习惯和工作环境。

考虑到设计等方面的因素，我们还可以认为软件缺陷还可以包括软件设计不符合规范，未能在特定的条件（资金、范围等）达到最佳等。可惜的是，我们中的很多人更倾向于把软件缺陷看成运行时出现问题上来，认为软件测试仅限于程序提交之后。

在目前的国内环境下，我们几乎看不到完整准确的客户需求说明书，加以客户的需求时时在变，追求完美的测试变得不太可能。因此作为一个优异的测试人员，追求软件质量的完美固然是我们的宗旨，但是明确软件测试现实与理想的差距，在软件测试中学会取舍和让步，对软件测试是有百益而无一弊的。

下面是一些软件测试的常识，对这些常识的理解和运用将有助于我们在进行软件测试时能够更好的把握软件测试的尺度。

- 测试是不完全的（测试不完全）

很显然，由于软件需求的不完整性、软件逻辑路径的组合性、输入数据的大量性及结果多样性等因素，哪怕是一个极其简单的程

序，要想穷尽所有逻辑路径，所有输入数据和验证所有结果是非常困难的一件事情。我们举一个简单的例子，比如说求两个整数的最大公约数。其输入信息为两个正整数。但是如果我们将整个正整数域的数字进行一番测试的话，从其数目的无限性我们便可证明是这样的测试在实际生活中是行不通的，即便某一天我们能够穷尽该程序，只怕我们乃至我们的子孙都早已作古了。为此作为软件测试，我们一般采用等价类和边界值分析等措施来进行实际的软件测试，寻找最小用例集合成为我们精简测试复杂性的一条必经之道。

- 测试具有免疫性（软件缺陷免疫性）

软件缺陷与病毒一样具有可怕的“免疫性”，测试人员对其采用的测试越多，其免疫能力就越强，寻找更多软件缺陷就更加困难。由数学上的概率论我们可以推出这一结论。假设一个 50000 行的程序中有 500 个软件缺陷并且这些软件错误分布时均匀的，则每 100 行可以找到一个软件缺陷。我们假设测试人员用某种方法花在查找软件缺陷的精力为 X 小时 /100 行。照此推算，软件存在 500 个缺陷时，我们查找一个软件缺陷需要 X 小时，当软件只存在 5 个错误时，我们每查找一个软件缺陷需要 100X 小时。实践证明，实际的测试过程比上面的假设更为苛刻，为此我们必须更换不同的测试方式和测试数据。该例子还说明了在软件测试中采用单一的方法不能高效和完全的针对所有软件缺陷，因此软件测试应该尽可能的多采用多种途径进行测试。

- 测试是“泛型概念”（全程测试）

我一直反对软件测试仅存在于程序完成之后。如果单纯的只将程序设计阶段后的阶段称之为软件测试的话，需求阶段和设计阶段的缺陷产生的放大效应会加大。这非常不利于保证软件质量。需求缺陷、设计缺陷也是软件缺陷，记住“软件缺陷具有生育能力”。软件测试应该跨越整个软件开发流程。需求验证（自检）和设计验证（自检）也可以算作软件测试（建议称为：需求测试和设计测试）的一种。软件测试应该是一个泛型概念，涵盖整个软件生命周期，

这样才能确保周期的每个阶段禁得起考验。同时测试本身也需要有第三者进行评估（信息系统审计和软件工程监理），即测试本身也应当被测试，从而确保测试自身的可靠性和高效性。否则自身不正，难以服人。

另外还需指出的是软件测试是提高软件产品质量的必要条件而非充分条件，软件测试是提高产品质量最直接、最快捷的手段，但决不是一个根本手段。

- 80-20 原则

80% 的软件缺陷常常生存在软件 20% 的空间里。这个原则告诉我们，如果你想使软件测试有效地话，记住常常光临其高危多发“ 地段 ” 。在那里发现软件缺陷的可能性会大的多。这一原则对于软件测试人员提高测试效率及缺陷发现率有着重大的意义。聪明的测试人员会根据这个原则很快找出较多的缺陷而愚蠢的测试人员却仍在漫无目的地到处搜寻。

80-20 原则的另外一种情况是，我们在系统分析、系统设计、系统实现阶段的复审，测试工作中能够发现和避免 80% 的软件缺陷，此后的系统测试能够帮助我们找出剩余缺陷中的 80% ，最后的 5% 的软件缺陷可能只有在系统交付使用后用户经过大范围、长时间使用后才会曝露出来。因为软件测试只能够保证尽可能多地发现软件缺陷，却无法保证能够发现所有的软件缺陷。

80-20 原则还能反映到软件测试的自动化方面上来，实践证明 80% 的软件缺陷可以借助人工测试而发现， 20% 的软件缺陷可以借助自动化测试能够得以发现。由于这二者间具有交叉的部分，因此尚有 5% 左右的软件缺陷需要通过其他方式进行发现和修正。

- 为效益而测试

为什么我们要实施软件测试，是为了提高项目的质量效益最终以提高项目的总体效益。为此我们不难得出我们在实施软件测试应该掌握的度。软件测试应该在软件测试成本和软件质量效益两者间找到一个平衡点。这个平衡点就是我们在实施软件测试时应该遵守

的度。单方面的追求都必然损害软件测试存在的价值和意义。一般说来，在软件测试中我们应该尽量地保持软件测试简单性，切勿将软件测试过度复杂化，拿物理学家爱因斯坦的话说就是： Keep it simple but not too simple 。

- 缺陷的必然性

软件测试中，由于错误的关联性，并不是所有的软件缺陷都能够得以修复。某些软件缺陷虽然能够得以修复但在修复的过程中我们会难免引入新的软件缺陷。很多软件缺陷之间是相互矛盾的，一个矛盾的消失必然会引发另外一个矛盾的产生。比如我们在解决通用性的缺陷后往往会带来执行效率上的缺陷。更何况在缺陷的修复过程中，我们常常还会受时间、成本等方面的限制因此无法有效、完整地修复所有的软件缺陷。因此评估软件缺陷的重要度、影响范围，选择一个折中的方案或是从非软件的因素（比如提升硬件性能）考虑软件缺陷成为我们在面对软件缺陷时一个必须直面的事实。

- 软件测试必须有预期结果

没有预期结果的测试是不可理喻的。软件缺陷是经过对比而得出来的。这正如没有标准无法进行度量一样。如果我们事先不知道或是无法肯定预期的结果，我们必然无法了解测试正确性。这很容易然人感觉如盲人摸象一般，不少测试人员常常凭借自身的感觉去评判软件缺陷的发生，其结果往往是把似是而非的东西作为正确的结果来判断，因此常常出现误测的现象。

- 软件测试的意义 - 事后分析

软件测试的目的单单是发现缺陷这么简单吗？如果是 “ 是 ” 的话，我敢保证，类似的软件缺陷在下一次新项目的软件测试中还会发生。古语说得好， “ 不知道历史的人必然会重蹈覆辙 ” 。没有对软件测试结果进行认真的分析，我们就无法了解缺陷发生的原因和应对措施，结果是我们不得不耗费的大量的人力和物力来再次查找软件缺陷。很可惜，目前大多测试团队都没有意识到这一点，测试报告中缺乏测试结果分析这一环节。

结论：

软件测试是一个需要“自觉”的过程，作为一个测试人员，遇事沉着，把持尺度，从根本上应对软件测试有着正确的认识，希望本文对读者对软件测试的认识有所帮助。

软件测试过程管理实践

作者：杨静波 田悦峰

摘 要： 随着测试技术的蓬勃发展，测试过程的管理显得犹为重要，过程管理已成为测试成功的重要保证。经过多年努力，测试专家提出了许多测试过程模型，包括 V 模型、W 模型、H 模型等等。这些模型定义了测试活动的流程和方法，为测试管理工作提供了指导。但这些模型各有长短，并没有哪种模型能够完全适合于所有的测试项目，在实际测试中应该吸取各模型的长处，归纳出合适的测试理念。“尽早测试”、“全面测试”、“全过程测试”和“独立、迭代的测试”是从各模型中提炼出来的四个理念，这些思想在实际测试项目中得到了应用并收到了良好的效果。在运用这些理念指导测试的同时，测试组应不断关注于基于度量和分析的过程的改进活动，不断提高测试管理水平，更好的提高测试效率、降低测试成本。

关键词： 测试过程模型；测试管理理念；可持续改进

引言

1963 年，在美国发生了这样一件事：编程人员把一个 FORTRAN 程序的循环语句 DO 5 I=1，3 误写为 DO 5 I=1.3。一点之差导致飞往火星的火箭爆炸，造成 1000 多万美元的损失。这种情况的发生，迫使人们考虑在软件投入使用之前必须进行彻底的测试。今天，在软件比较发达的国家，软件测试已经成为一个独立的产业，软件公司纷纷建立独立的测试队伍研究测试技术并开展测试工作。中国的软件测试起步较晚，但随着我国软件产业的蓬勃发展以及人们对软件质量的重视，软件测试正在成为一个新兴的产业。近两年来，国内新成立专业性测试机构 10 余家，一批批专业的软件测试人员正涌现出来。每年国内都有大量的测试技术交流会议举办，有大量的测

试研究论文在专业刊物上发表。在测试技术发展的同时，测试过程的管理显得犹为重要。一个成功的测试项目，离不开对测试过程科学的组织和监控，过程管理已成为测试成功的重要保证。

1 测试过程概述

1.1 软件测试过程概述

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。众所周知，开发过程的质量决定了软件的质量，同样的，测试过程的质量将直接影响测试结果的准确性和有效性。软件测试过程和软件开发过程一样，都遵循软件工程原理，遵循管理学原理。

随着测试过程管理的发展，软件测试专家通过实践总结出了很多很好的测试过程模型。这些模型将测试活动进行了抽象，并与开发活动有机的进行了结合，是测试过程管理的重要参考依据。

1.2 软件测试过程模型介绍

V 模型

V 模型最早是由 Paul Rook 在 20 世纪 80 年代后期提出的，旨在改进软件开发的效率和效果。V 模型反映出了测试活动与分析设计活动的关系。在图 1-1 中，从左到右描述了基本的开发过程和测试行为，非常明确的标注了测试过程中存在的不同类型的测试，并且清楚的描述了这些测试阶段和开发过程期间各阶段的对应关系。

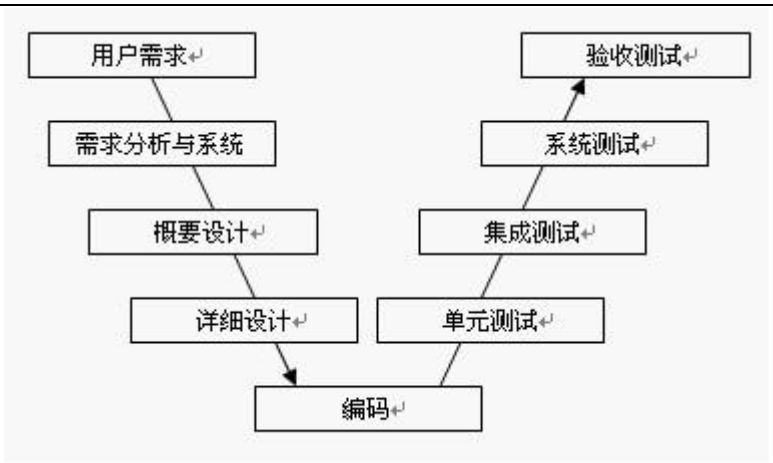


图 1-1　软件测试 V 模型

V 模型指出，单元和集成测试应检测程序的执行是否满足软件设计的要求；系统测试应检测系统功能、性能的质量特性是否达到系统要求的指标；验收测试确定软件的实现是否满足用户需要或合同的要求。

但 V 模型存在一定的局限性，它仅仅把测试作为在编码之后的一个阶段，是针对程序进行的寻找错误的活动，而忽视了测试活动对需求分析、系统设计等活动的验证和确认的功能。

W 模型

W 模型由 Evolutif 公司公司提出，相对于 V 模型，W 模型增加了软件各开发阶段中应同步进行的验证和确认活动。如图 1-2 所示，W 模型由两个 V 字型模型组成，分别代表测试与开发过程，图中明确表示出了测试与开发的并行关系。

W 模型强调：测试伴随着整个软件开发周期，而且测试的对象不仅仅是程序，需求、设计等同样要测试，也就是说，测试与开发是同步进行的。W 模型有利于尽早地全面的发现问题。例如，需求分析完成后，测试人员就应该参与到对需求的验证和确认活动中，以尽早地找出缺陷所在。同时，对需求的测试也有利于及时了解项

目难度和测试风险，及早制定应对措施，这将显著减少总体测试时间，加快项目进度。

但 W 模型也存在局限性。在 W 模型中，需求、设计、编码等活动被视为串行的，同时，测试和开发活动也保持着一种线性的前后关系，上一阶段完全结束，才可正式开始下一个阶段工作。这样就无法支持迭代的开发模型。对于当前软件开发复杂多变的情况，W 模型并不能解除测试管理面临着困惑。

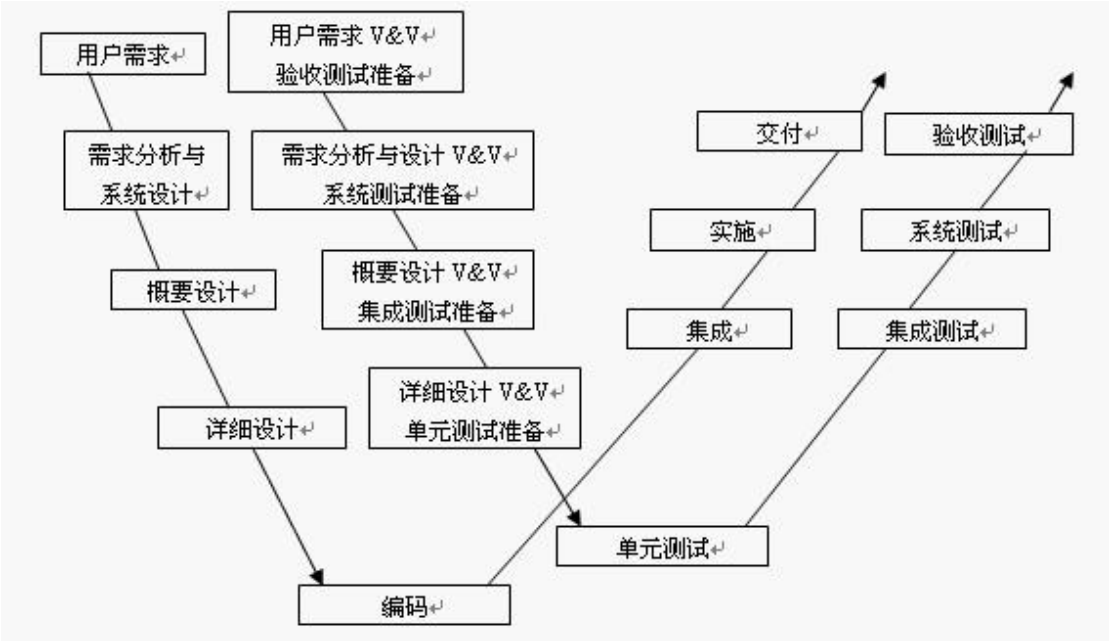


图 1-2　软件测试 W 模型

H 模型

V 模型和 W 模型均存在一些不妥之处。如前所述，它们都把软件的开发视为需求、设计、编码等一系列串行的活动，而事实上，这些活动在大部分时间内是可以交叉进行的，所以，相应的测试之间也不存在严格的次序关系。同时，各层次的测试（单元测试、集成测试、系统测试等）也存在反复触发、迭代的关系。

为了解决以上问题，有专家提出了 H 模型。它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 1-3 所示。

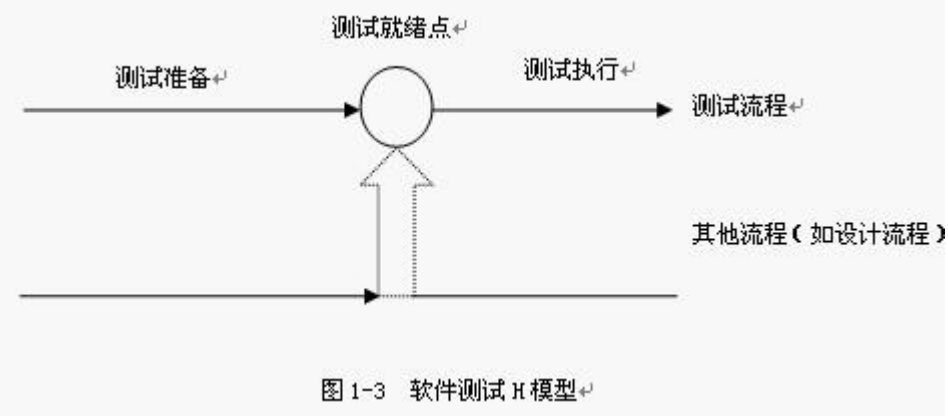


图 1-3 软件测试 H 模型

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。

H 模型揭示了一个原理：软件测试是一个独立的流程，贯穿产品整个生命周期，与其他流程并发地进行。H 模型指出软件测试要尽早准备，尽早执行。不同的测试活动可以是按照某个次序先后进行的，但也可能是反复的，只要某个测试达到准备就绪点，测试执行活动就可以开展。

其他模型

除上述几种常见模型外，业界还流传着其他几种模型，例如 X 模型、前置测试模型等。X 模型提出针对单独的程序片段进行相互

分离的编码和测试，此后通过频繁的交接，通过集成最终合成为可执行的程序。前置测试模型体现了开发与测试的结合，要求对每一个交付内容进行测试。这些模型都针对其他模型的缺点提出了一些修正意见，但本身也可能存在一些不周到的地方。所以在测试过程管理中，正确选取过程模型是一个关键问题。

1.3 软件测试过程模型选取策略

前面介绍的测试过程模型中，V 模型强调了在整个项目开发中需要经历的不同的测试级别，但忽视了测试的对象不应该仅仅是程序。而 W 模型在这一点上进行了补充，明确指出应该对需求、设计进行测试。但是 V 模型和 W 模型都没有将一个完整的测试过程抽象出来，成为一个独立的流程，这并不适合当前软件开发中广泛应用的迭代模型。H 模型则明确指出测试的独立性，也就是说只要测试条件成熟了，就可以开展测试。

在实际测试工作中我们应该尽可能地去应用各模型中对项目有实用价值的方面，不能强行的为使用模型而使用模型。在测试实践中，我们采用的方法是：以 W 模型作为框架，及早的、全面的开展测试。同时灵活运用 H 模型独立测试的思想，在达到恰当的就绪点时就应该开展独立的测试工作，同时将测试工作进行迭代，最终保证完成测试目标。

2 测试过程管理理念

生命周期模型为我们提供了软件测试的流程和方法，为测试过程管理提供了依据。但实际的测试工作是复杂而烦琐的，可能不会有哪种模型完全适用于某项测试工作。所以，我们应该从不同的模型中抽象出符合实际现状的测试过程管理理念，依据这些理念来策

划测试过程，以不变应万变。当然测试管理牵涉的范围非常的广泛，包括过程定义、人力资源管理、风险管理等等，本节仅介绍几条从过程模型中提炼出来的，对实际测试有指导意义的管理理念。

3 尽早测试

“尽早测试”是从 W 模型中抽象出来的理念。我们说测试并不是在代码编写完成之后才开展的工作，测试与开发是两个相互依存的并行的过程，测试活动在开发活动的前期已经开展。

“尽早测试”包含两方面的含义：第一，测试人员早期参与软件项目，及时开展测试的准备工作，包括编写测试计划、制定测试方案以及准备测试用例；第二，尽早的开展测试执行工作，一旦代码模块完成就应该及时开展单元测试，一旦代码模块被集成成为相对独立的子系统，便可以开展集成测试，一旦有 BUILD 提交，便可以开展系统测试工作。

由于及早的开展了测试准备工作，测试人员能够于早期了解测试的难度、预测测试的风险，从而有效提高了测试效率，规避测试风险。由于及早的开展测试执行工作，测试人员尽早的发现软件缺陷，大大降低了 BUG 修复成本。但是需要注意，“尽早测试”并非盲目的提前测试活动，测试活动开展的前提是达到必须的测试就绪点。

4 全面测试

软件是程序、数据和文档的集合，那么对软件进行测试，就不仅仅是对程序的测试，还应包括软件“副产品”的“全面测试”，这是 W 模型中一个重要的思想。需求文档、设计文档作为软件的阶段性产品，直接影响到软件的质量。阶段产品质量是软件质量的量的积累，不能把握这些阶段产品的质量将导致最终软件质量的不可

控。

“全面测试”包含两层含义：第一，对软件的所有产品进行全面的测试，包括需求、设计文档，代码，用户文档等等。第二，软件开发及测试人员（有时包括用户）全面的参与到测试工作中，例如对需求的验证和确认活动，就需要开发、测试及用户的全面参与，毕竟测试活动并不仅仅是保证软件运行正确，同时还要保证软件满足了用户的需求。

“全面测试”有助于全方位把握软件质量，尽最大可能的排除造成软件质量问题的因素，从而保证软件满足质量需求。

5 全过程测试

在 W 模型中充分体现的另一个理念就是“全过程测试”。双 V 字过程图形象的表明了软件开发与软件测试的紧密结合，这就说明软件开发和测试过程会彼此影响，这就要求测试人员对开发和测试的全过程进行充分的关注。

“全过程测试”包含两层含义：第一，测试人员要充分关注开发过程，对开发过程的各种变化及时做出响应。例如开发进度的调整可能会引起测试进度及测试策略的调整，需求的变更会影响到测试的执行等等。第二，测试人员要对测试的全过程进行全程的跟踪，例如建立完善的度量与分析机制，通过对自身过程的度量，及时了解过程信息，调整测试策略。

“全过程测试”有助于及时应对项目变化，降低测试风险。同时对测试过程的度量与分析也有助于把握测试过程，调整测试策略，便于测试过程的改进。

6 独立的、迭代的测试

我们知道，软件开发瀑布模型只是一种理想状况。为适应不同

的需要，人们在软件开发过程中摸索出了如螺旋、迭代等诸多模型，这些中需求、设计、编码工作可能重叠并反复进行的，这时的测试工作将也是迭代和反复的。如果不能将测试从开发中抽象出来进行管理，势必使测试管理陷入困境。

软件测试与软件开发是紧密结合的，但并不代表测试是依附于开发的一个过程，测试活动是独立的。这正是 H 模型所主导的思想。“独立的、迭代的测试”着重强调了测试的就绪点，也就是说，只要测试条件成熟，测试准备活动完成，测试的执行活动就可以开展。

所以，我们在遵循尽早测试、全面测试、全过程测试理念的同时，应当将测试过程从开发过程中适当的抽象出来，作为一个独立的过程进行管理。时刻把握独立的、迭代测试的理念，减小因开发模型的繁杂给测试管理工作带来的不便。对于软件过程中不同阶段的产品和不同的测试类型，只要测试准备工作就绪，就可以及时开展测试工作，把握产品质量。

7 测试过程管理实践

本节以一个实际项目系统测试过程（不对单元测试和集成测试过程进行分析）的几个关键过程管理行为为例，来阐述上节中提出的测试理念。在一个构件化 ERP 项目中，由于前期需求不明确，开发周期相对较长，为了对项目进行更好的跟踪和管理，项目采用增量和迭代模型进行开发。整个项目开发共分三个阶段完成：第一阶段实现进销存的简单的功能和工作流；第二阶段：实现固定资产管理、财务管理，并完善第一阶段的进销存功能；第三阶段：增加办公自动化的管理（OA）。该项目每一阶段工作是对上一阶段成果的一次迭代完善，同时将新功能进行了一次叠加。

8 策划测试过程

依据传统的方法，将系统测试作为软件开发的一个阶段，系统测试执行工作将在三个阶段完成后开展，很明显，这样做不利于 BUG 的及时暴露。有些缺陷可能会埋藏至后期发现，这时的修复成本将大大提高。我们依据“独立和迭代”的测试理念，在本系统中，对测试过程进行独立的策划，找出测试准备就绪点，在就绪点及时开展测试。该系统的三个阶段具有相对的独立性，在每一阶段完成所提交的阶段产品具有相对的独立性，可以作为系统测试准备的就绪点。故而，在该系统开发过程中，系统测试组计划开展三阶段的系统测试，每个阶段系统测试具有不同的侧重点，目的在于更好的配合开发工作尽早发现软件 BUG，降低软件成本。软件开发与系统测试过程的关系如图 3-1 所示。

实践证明，这种做法起到了预期的效果，与开发过程紧密结合而又相对独立的测试过程，有效的于早期发现了许多系统缺陷，降低了开发成本，同时也使基于复杂开发模型的测试管理工作更加清晰明了。

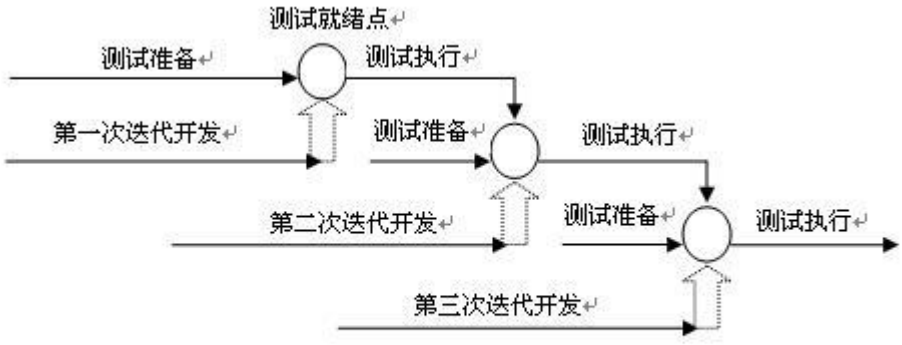


图 3-1　软件开发与系统测试关系图

9　把握需求

在本系统开发过程中，需求的获取和完善贯穿每个阶段。对需求的把握很大程度上决定了软件测试是否能够成功。系统测试不仅仅确认软件是否正确实现功能，同时还要确认软件是否满足用户的

需要。依据“尽早测试”和“全面测试”原则，在需求的获取阶段，测试人员参与到了对需求的讨论之中。测试人员与开发人员及用户一起讨论需求的完善性与正确性，同时从可测试性角度为需求文档提出建议。这些建议对开发人员来说，是从一个全新的思维角度提出的约束。同时，测试组结合前期对项目的把握，很容易制定出了完善的测试计划和方案，将各阶段产品的测试方法及进度、人员安排进行了策划，使整个项目的进展有条不紊。

实践证明，测试人员早期参与需求的获取和分析中，有助于加深测试人员对需求的把握和理解，同时也大大促进需求文档的质量。在需求人员把握需求的同时，于早期制定项目计划和方案，及早准备测试活动，大大提高了测试效率。

10 变更控制

变更控制体现的是“全过程测试”理念。在软件开发过程中，变更往往是不可避免的，变更也是造成软件风险的重要因素。在本系统测试中，仅第一阶段就发生了 7 次需求变更，调整了两次进度计划。依据“全过程测试”理念，测试组密切关注开发过程，跟随进度计划的变更调整测试策略，依据需求的变更及时补充和完善测试用例。由于充分的测试准备工作，在测试执行过程中，没有废弃一个测试用例，测试的进度并没有因为变更而受到过多影响。

11 度量与分析

对测试过程的度量与分析同样体现的“全过程测试”理念。对测试过程的度量有利于及时把握项目情况，对过程数据进行分析，很容易发现优势劣势，找出需要改进的地方，及时调整测试策略。

在 ERP 项目中，我们在测试过程中对不同阶段的 BUG 数量进行了度量，并分析测试执行是否充分。如图 3-2 所示，通过分析我们

得出：相同时间间隔内发现的 BUG 数量呈收敛状态，测试是充分的。在 BUG 数量收敛的状态下结束细测是恰当的。

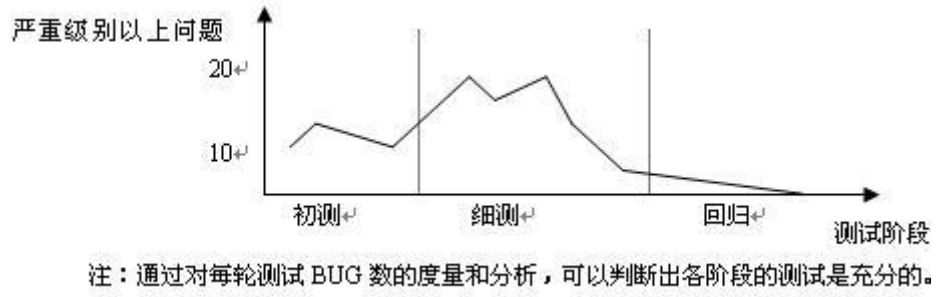


图 3-2　软件开发与系统测试关系图

测试中，我们对不同功能点的测试数据覆盖率和发现问题数进行度量，以便分析测试用例的充分度与 BUG 发现率之间的关系。如表 3-1 所示，对类似模块进行对比发现：某一功能点上所覆盖的测试数据组越多，BUG 的用例发现率越高。如果再结合工作量、用例执行时间等因素进行统计分析，便可以找到适合实际情况的测试用例书写粒度，从而帮助测试人员判断测试成本和收益间的最佳平衡点。

表 3-1 测试数据覆盖率与 BUG 发现率对应表

模块名称	功能点数	测试数据数	测试数据覆盖率	BUG 的用例发现率（）
模块 AA	6 个	75 组	12.5 组/每功能点	40%　（6/15）
模块 BB	30 个	96 组	3.3 组/每功能点	17%　（7/42）
模块 CC	15 个	87 组	5.1 组/每功能点	18%　（5/28）
模块 DD	16 个	46 组	2.8 组/每功能点	23%　（5/22）
…	…	…	…	…

注：通过统计可以得出测试数据与 BUG 发现率之间的关系，便于及时调整测试用例编写策略。

所有这些度量都是对测试全过程进行跟踪的结果，是及时调整测试策略的依据。对测试过程的度量与分析能有效的提高了测试效率，降低了测试风险。同时，度量与分析也是软件测试过程可持续改进的基本。

12 测试过程可持续改进

测试技术发展到今天，已经存在诸多可供参考的测试过程管理思想和理念。但信息技术发展一日千里，新技术不断涌现，这就注定测试过程也需要不断的改进。我们提倡基于度量与分析的可持续过程改进方法（本文不做详细论述）。在这种方法中，对现状的度量被制度化，并作为过程改进的基础。组织可以自定义需要度量的过程数据，将收集来的数据加以分析，以找出需要改进的因素。在不断的改进中，同步的调整需要度量的过程数据，使度量与分析始终为了过程改进服务，而过程改进也包含对度量和分析的改进。

掌握了基于度量和分析的可持续过程改进方法，测试过程管理将能够不断完善，测试活动将能够始终处于优化状态。

参考文献：

[1] 郑人杰，殷人昆．实用软件工程．第二版．北京：清华大学出版社，1997：203．

[2] 柳纯录，黄子河，陈渌萍．软件评测师教程．北京：清华大学出版社，2005：92．

[3] 林锐，等．CMMI3级软件过程改进方法与规范．北京：电子工业出版社，2003：119．

[4] 中华人民共和国国家标准.评价者用的过程.GB/T18905.

论测试人员为什么需要参加需求评审

作者：王树文

摘要：测试人员越早介入项目工作越好的观点已经被越来越多的测试人员所接受。在软件生命周期中，越晚发现的错误越难修改，修改成本越昂贵的论断也已经成为了大家的共识。

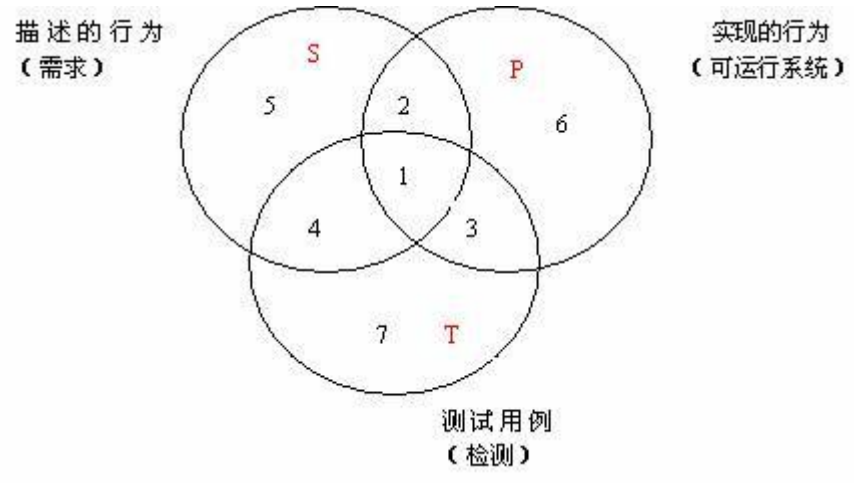
测试人员需要参加需求评审，我想大部分测试人员都接受了这个观点，同时也是这么做的。但测试人员为什么需要参加需求评审，恐怕不是每一个人都知道个中道理。本文从作者多年从事软件测试和过程管理的经验出发进行论述，供同行们参考。

关键词：测试；测试人员；需求评审

我们知道，在项目的生命周期中，需求、设计、编码、测试等活动往往是由不同的专业技术人员协同完成的。这样，由于下游技术人员对上游技术人员工作产出物的理解偏差，将导致不同阶段的产出物之间不一致的现象出现，这也是导致项目可能不成功的重要风险之一。

基于以上的观点，需求人员编写的《用户需求说明书》、系统设计人员编写的《系统设计说明书》、编码人员实现的系统、测试人员编写的《测试用例》之间不可避免地存在不一致的现象。

由于测试工作的主要面向对象是《用户需求说明书》和可运行系统，为便于分析，我们先用图来表述一下三者之间的关系：



从理论上讲，上图中“需求”（S 圆表示）、“可运行系统” （P 圆表示）和“测试用例” （T 圆表示）应该是重合，但实际上这三个圆很难重合。

从上图可以看出：可能会有没有测试到的已描述行为（区域 2 和区域 5）；经过测试的已描述行为（区域 1 和区域 4）；对应未描述行为的测试用例（区域 3 和区域 7）；可能会没有测试的程序行为（区域 2 和区域 6）；经过测试的程序行为（区域 1 和区域 3）；对应于未通过程序实现的行为（区域 4 和区域 7）。

由于我们今天讨论的主题是“测试人员为什么需要参加需求评审”，因此我们重点来看看与这一问题关系密切的区域 2 和区域 5、区域 3 和区域 7。

区域 2 和区域 5 这两部分是测试用例没有覆盖到的需求，即“没有正确的测试用例与该部分需求对应”。出现这种情况本人认为有三种可能：

1. 设计测试用例的测试人员对需求理解不完整，出现了应该被测试的需求而没有被测试用例覆盖到的现象；

2. 《用户需求说明书》中区域 2 和区域 5 对应的需求，具有不可测试性，测试人员无法设计用例；

3. 《用户需求说明书》中区域 2 和区域 5 对应的需求，不是用户的需求，是需求分析人员凭空增加的，测试人员无须设计与这部分需求对应的测试用例。

区域3和区域7这两部分是需求没有覆盖到的测试用例，即“没有正确的需求与该部分测试用例相对应”。出现这种情况本人认为有两种可能：

1、 设计测试用例的测试人员对需求理解不充分，从而设计出了多余的测试用例；

2、 《测试用例》中区域3和区域7的测试用例所对应的需求，是用户的真实需求，只是《用户需求说明书》中没有描述到而已。

从上面的分析可以看出，如果测试人员参加了“需求评审”，则上面出现的问题可以最大程度地避免，因为测试人员参加“需求评审”的重要作用正是针对产生这些问题的因素而进行的。

由此总结出，测试人员参加“需求评审”活动所需要达到的目标包括如下三个方面：

1. 充分地理解需求，确保对需求的理解与需求分析人员是一致的；

2. 从可测试的角度，努力发现《用户需求说明书》中不可测试的需求，从而提醒需求分析人员尽早修改；

3. 从测试人员的角度努力发现《用户需求说明书》中的不完整性，从而提醒需求分析人员及时补充遗漏掉的这部分用户需求。

参考文献

[1] 《软件测试》 机械工业出版社

[2] 《软件测试经验与教训》（美）Cem Kaner,James Bach,Bret Pettichord 著，韩柯 等译

[3]《编写有效用例》（美）Alistair Cockburn 著，王雷、张莉译 机械工业出版社

深入理解 LoadRunner 测试结果

作者: 李康

本文所面向的主要是对 LoadRunner 的性能数据采集有深入了解的应用性能工程师，或者希望把 LoadRunner 的测试结果集成到其企业自身的数据表现平台的开发人员。

在阅读本文之前，确认您已经具备以下的相关知识：

1. LoadRunner 的架构和测试执行流程

2. LoadRunner 脚本开发和交易类型

3. LoadRunner 实时监控器和相关设定

4. LoadRunner 交易类数据收集和相关设定

5. LoadRunner 用户性能数据收集方法

假设

1. 为了方便说明，作者假设 LoadRunner 的测试结果已被存放在了 C:\Sample\Results 目录中，并被取名为 Result1。
2. 同样 LoadRunner 的安装目录为 C:\Mercury\LoadRunner，在下文中以%LR%说明。

测试场景信息

显然，LoadRunner 测试场景的结果将被存放在一个目录中。

这个目录就是以测试结果的名字而命名：

RESULT_DIR＝C:\Sample\Results\Result1

测试结果的主文件名为：

RESULT_FILE="C:\Sample\Results\Result1\Result1.lrr"

所有的 LoadRunner 测试结果的主文件都是以.lrr 为文件扩展名的，这个文件是一个类 INI 文件的格式，其中包括了测试场景执行时的重要信息：

[Scenario]

Time_Zome="#"

Start_time="#"

Daylight_Bias="#"

Stop_time="#"

下面是一个具体的例子：

[Scenario]

Time_Zone="28800"

Start_time="1075066329"

Daylight_Bias="0"

Stop_time="1075066555"

请注意其中时间都是以 C 语言的 time_t 所兼容的长整型所定

义，即从 1970 年 1 月 1 日 0 点以来的秒数，在这里表示为测试开始时间是：2004 年 1 月 25 日 13:32:09，测试结束时间是：2004 年 1 月 25 日 13:35:55。

虚拟用户交易性能统计

在 LoadRunner 的测试结果中，所有的虚拟用户的执行结果都被保存在了以.EVE 为扩展名的文件中。这些文件被列举在了 Remote_results.txt 中（如本文的例子即为 C:\Sample\Results\Result1\remote_results.txt ）。关于 Remote_result.txt 的文件格式为：

[Hosts]

=""

=""

…

=""

下面是一个实例：

[Hosts]

localhost="C:\Sample\Results\Result1\localhost_1.eve"

ibmt40= C:\Sample\Results\Result1\ibmt40_1.eve

所有的.EVE 文件被存放在了%RESULT_DIR%中，比如上面的例子中，表示在“Localhost”上执行的虚拟用户结果被保存在了%RESULTDIR%\localhost_1.eve 中。这些文件是以二进制形式存放的。在 LoadRunner 的预处理过程中,它们将会被转换成平文本格式。同样可以利用在本文中所附的工具做这样的转换，具体过程如下：

安装转换工具（笔者在 LoadRunner 7.8 环境中验证通过，不确定在 8.0 及其后版本中的可用性）

1. 拷贝相关文件至%LR%所对应的目录中

2. 转换 EVE 文件为文本格式

通过命令行方式（CMD.EXE）执行 %LR%/BIN 下的 cmd_reader_eve_file，命令行格式如下：

cmd_reader_eve_file

具体实例如下：

Cmd_reader_eve_file C:\Sample\Results\Result1\localhost1.eve C:\Sample\Result\Result1\eve_output 1 0

值得注意的是，LoadRunner 没有对其中的参数（Flag）作出解释，在此处的 VERTICAL_FLAG 为 0 表示每一数据行对应一个交易（Transaction）或一个数据点（Data Point）入口。

作为 cmd_reader_eve_file 的输出，LoadRunner 将在目录中产生 eve_text.log 和 read_eve.log 文件：

a) eve_text.log 包括了所有在输入的.EVE 文件中的交易性能统计数据或数据点信息。

b) read_eve.log 则是 cmd_reader_eve_file 的执行日志，包括了其状态。

3. 进一步理解 eve_text.log 文件

该文件的基本格式可能有 2 种，并且这 2 种格式都是有效格式。

格式 1：

各字段的意义如下：

=1 固定，交易数据类型

交易开始时间

交易结束时间

交易名称

交易句柄

父交易句柄

总思考时间（由用户定义的 lr_thing_time 或 sleep 函数产生）

总耗费时间（用于 LoadRunner 生成请求和处理响应时间等耗

损时间）

交易状态，正常结束时为 0

虚拟用户 ID

用户组名

测试场景 ID

测试脚步名称

交易类型，其中 1 为用户定义交易，2 为自动交易，3 为脚本模块或 LoadRunner 函数交易

其中典型的交易响应时间为：

交易时间＝交易结束时间－交易开始时间－总思考时间－总耗费时间

i. 由 lr_start_transaction/lr_end_transaction 函数生成的交易的交易类型＝1

ii. 由脚本 API 定义或通过运行时设定的交易的交易类型＝2

iii. 由脚本或 LoadRunner 函数所定义交易（如 vuser_init,vuser_end 等）的交易类型="3"

以下是这类数据格式的实例：

1 1075066330.631081 1075066330.631413 vuser_init_Transaction 1,1 0,0 0.000000 0.000000 0 1 sample_user 1 sample_user 3

1 1075066330.636429 1075066331.944108 LOGON 1,3 1,2 1.294478 0.000000 0 1 sample_user 1 sample_user 1

1 1075066340.345697 1075066340.348619 SUBMIT_REQUEST 1,12 1,2 0.000000 0.000000 0 1 sample_user 1 sample_user 1

1 1075066343.224992 1075066343.227754 homepage_Action_21 1,16 1,15 0.000000 0.000000 0 1 sample_user 1 sample_user 2

1 1075066343.224884 1075066343.227792 LOGOFF 1,15 1,2 0.000000 0.000000 0 1 sample_user 1 sample_user 1

1 1075066330.636378 1075066343.227842 Action_Transaction 1,2 0,0 12.572037 0.000000 0 1 sample_user 1 sample_user 3

格式 2：

各字段的意义如下表

=2 固定，数据点类型

数据点采集的时间戳

数据点名称

数据点句柄

父数据句柄

数据点值

虚拟用户 ID

用户组名称

测试场景 ID

脚本名称

数据点类型

数据点可以有 LoadRunner 的其他协同模块收集，也可以由比如 lr_user_data_point 等函数产生。

以下是该格式的实例：

2 1075066343.227884 HTTP_200 1,18 0,0 2.000000 1 sample_user 1 sample_user 1

2 1075066345.350112 mic_recv 1,19 0,0 14705.000000 1 sample_user 1 sample_user 1

服务器监控性能统计

在 LoadRunner 中，几乎所有的服务器信息是通过 LoadRunner Realtime Monitor 来采集的。这些采集的信息最终将被存放在%RESULT_DIR%\offline.dat 中。

offline.data 的一般格式为：

具体实例如下：

offline_datapoint_0 1075066332 69705728.000

offline_datapoint_1 1075066330 4094.388

offline_datapoint_2 1075066330 412168.350

offline_datapoint_11 1075066330 22.000

offline_datapoint_3 1075066330 0.000

这些数据点所代表的实际监控指标被定义在了统一目录（%RESULT_DIR%）中的各自的 offl_*.def 文件中。每个 offl_*.def 都代表着一个性能指标。Offl__*.def 是一个 INI 格式的文件，其中

的[Graph definition]定义块详细描述了该性能指标。

GraphGroupMenuTitle

性能图表（组）的名称

GraphTitle

性能指标名称

XAxisTitle

X 轴名称

YAxisTitle

Y 轴名称

XAxisIsElaspsedTime

TRUE/FALSE,X 轴是否为时间

LineType

NOSTEP,固定

BuildUnderLoadGraph

TRUE/FALSE

AggregateFunction

汇总类型，如 SUM，AVG 等

MissingData

GranularityMode

粒度模型，比如 NOTINCLUDEZERO

NoDataBehavior

GraphType

Count=#

数据点数量

DataPointLabel_#

数据点别名

LineTitle_#

Host 名称，数据点源

LineGroup_#

DataPointDescr_#

数据点的描述

下面是一个 offl_*.def 文件的实例：

[Graph definition]

count="1"

GraphTitle="%" Processor Time (Process aspnet_wp)

DataPointLabel_1="offline_datapoint_3"

LineTitle_1="localhost"

LineGroup_1="0"

YAxisTitle="%" Processor Time (Process aspnet_wp)

GraphGroupMenuTitle="Resource" Monitoring

XAxisTitle="Elapsed" Scenario Time (seconds)

XAxisIsElapsedTime="TRUE"

LineType="NOSTEP"

BuildUnderLoadGraph="TRUE"

AggregateFunction="AVG"

MissingData="PREVIOUS"

GranularityMode="NOTINCLUDEZERO"

NoDataBehavior="RemoveGraph"

GraphType="es_rm_svr_res_nt"

有关 GraphType 的定义，请参考 LoadRunner 安装目录下的 lr_ext 目录中的相关信息。比如，es_rm_svr_res_nt 对应 Windows Resources；es_rm_svr_rstat_unix 对应 UNIX Resources 等。

深入浅出单元测试

作者：老纳

一、 单元测试概述

工厂在组装一台电视机之前，会对每个元件都进行测试，这，就是单元测试。

其实我们每天都在做单元测试。你写了一个函数，除了极简单的外，总是要执行一下，看看功能是否正常，有时还要想办法输出些数据，如弹出信息窗口什么的，这，也是单元测试，老纳把这种单元测试称为临时单元测试。只进行了临时单元测试的软件，针对代码的测试很不完整，代码覆盖率要超过 70%都很困难，未覆盖的代码可能遗留大量的细小的错误，这些错误还会互相影响，当 BUG 暴露出来的时候难于调试，大幅度提高后期测试和维护成本，也降低了开发商的竞争力。可以说，进行充分的单元测试，是提高软件质量，降低开发成本的必由之路。

对于程序员来说，如果养成了对自己写的代码进行单元测试的习惯，不但可以写出高质量的代码，而且还能提高编程水平。

多数讲述单元测试的文章都是以 Java 为例，本文以 C++为例，后半部分所介绍的单元测试工具也只介绍 C++单元测试工具。

要进行充分的单元测试，应专门编写测试代码，并与产品代码隔离。老纳认为，比较简单的办法是为产品工程建立对应的测试工程，为每个类建立对应的测试类，为每个函数（很简单的除外）建立测试函数。首先就几个概念谈谈老纳的看法。

一般认为，在结构化程序时代，单元测试所说的单元是指函数，在当今的面向对象时代，单元测试的所说的单元是指类。老纳认为，以类作为测试单位，复杂度高，可操作性较差，因此仍然应以函数

作为单元测试的测试单位，但可以用一个测试类来组织某个类的所有测试函数。单元测试不应过分强调面向对象，因为局部代码依然是结构化的。单元测试的工作量较大，简单实用高效才是硬道理。

有一种看法是，只测试类的接口(公有函数)，不测试其他函数，从面向对象角度来看，似乎有些道理。老纳认为，测试的目的是找错并最终排错，因此，只要是包含错误的可能性较大的函数都要测试，跟函数是否私有没有关系。对于 C++来说，可以用一种简单的方法区隔需测试的函数：简单的函数如数据读写函数的实现在头文件中编写(inline 函数)，所有在源文件编写实现的函数都要进行测试(构造函数和析构函数除外)。

什么时候测试？XP 开发理论讲究 TDD，即测试驱动开发，先编写测试代码，再进行开发。老纳认为，不必过分强调先什么后什么，重要的是高效和感觉舒适。从老纳的经验来看，先编写产品函数的框架，然后编写测试函数，针对产品函数的功能编写测试用例，然后编写产品函数的代码，每写一个功能点都运行测试，随时补充测试用例。所谓先编写产品函数的框架，是指先编写函数空的实现，有返回值的随便返回一个值，编译通后再编写测试代码，这时，函数名、参数表、返回类型都应该确定下来了，所编写的测试代码以后需修改的可能性比较小。

由谁测试？单元测试与其他测试不同，单元测试可看作是编码工作的一部分，应该由程序员完成，也就是说，经过了单元测试的代码才是已完成的代码，提交产品代码时也要同时提交测试代码。测试部门可以作一定程度的审核。

关于桩代码，老纳认为，单元测试应避免编写桩代码。桩代码就是用来代替某些代码的代码，例如，产品函数或测试函数调用了一个未编写的函数，可以编写桩函数来代替该被调用的函数，桩代码也用于实现测试隔离。采用由底向上的开发方式进行开发，先开发的代码先测试，可以避免编写桩代码，这样做的好处有：减少了工作量；测试上层函数时，也是对下层函数的间接测试；当下层函

数修改时，通过回归测试可以确认修改是否导致上层函数产生错误。

二、 测试代码编写

下面该说说如果编写测试代码了，以一个例子来说明，开发环境是 VC6.0。

产品类：

```
class CMyClass
{
public:
int Add(int i, int j);
CMyClass();
virtual ~CMyClass();

private:
int mAge; //年龄
CString mPhase; //年龄阶段，如"少年"，"青年"
};
```

建立对应的测试类 CMyClassTester，为了节约编幅，只列出源文件的代码：

```
void CMyClassTester::CaseBegin()
{
//pObj 是 CMyClassTester 类的成员变量，是被测试类的对象的指针，
//为求简单，所有的测试类都可以用 pObj 命名被测试对象的指针。
pObj = new CMyClass();
}
```

```
void CMyClassTester::CaseEnd()
{
delete pObj;
}
```

测试类的函数 CaseBegin()和 CaseEnd()建立和销毁被测试对象，每个测试用例的开头都要调用 CaseBegin()，结尾都要调用 CaseEnd()。

接下来，我们建立示例的产品函数：

```
int CMyClass::Add(int i, int j)
{
return i+j;
}
```

和对应的测试函数：

```
void CMyClassTester::Add_int_int()
{
}
```

把参数表作为函数名的一部分，这样当出现重载的被测试函数时，测试函数不会产生命名冲突。下面添加测试用例：

```
void CMyClassTester::Add_int_int()
{
//第一个测试用例
CaseBegin();{ //1
int i = 0; //2
int j = 0; //3
int ret = pObj->Add(i, j); //4
ASSERT(ret == 0); //5
}CaseEnd(); //6
}
```

第 1 和第 6 行建立和销毁被测试对象，所加的{}是为了让每个测试用例的代码有一个独立的域，以便多个测试用例使用相同的变量名。

第 2 和第 3 行是定义输入数据，第 4 行是调用被测试函数，这些容易理解，不作进一步解释。第 5 行是预期输出，它的特点是当实际输出与预期输出不同时自动报错，ASSERT 是 VC 的断言宏，也可以使用其他类似功能的宏，使用测试工具进行单元测试时，可以使用该工具定义的断言宏。

示例中的格式显得很不简洁，2、3、4、5 行可以合写为一行：ASSERT(pObj->Add(0, 0) == 0);但这种不简洁的格式却是老纳极力推荐的，因为它一目了然，易于建立多个测试用例，并且具有很好的适应性，同时，也是极佳的代码文档，总之，老纳建议：输入数据和预期输出要自成一块。

建立了第一个测试用例后，应编译并运行测试，以排除语法错误，然后，使用拷贝已有的测试用例并进行修改的办法建立其他测试用例。由于各个测试用例之间的差别往往很小，通常只需修改一两个数据，拷贝修改是最快的建立多个测试用例的办法。

三、 测试用例

下面说说测试用例、输入数据及预期输出。输入数据是测试用例的核心，老纳对输入数据的定义是：被测试函数所读取的外部数据及这些数据的初始值。外部数据是对于被测试函数来说的，实际上就是除了局部变量以外的其他数据，老纳把这些数据分为几类：参数、成员变量、全局变量、IO 媒体。IO 媒体是指文件、数据库或其他储存或传输数据的媒体，例如，被测试函数要从文件读取数据，那么，文件中的原始数据也属于输入数据。一个函数无论多复杂，都无非是对这几类数据的读取、计算和写入。预期输出是指：返回值及被测试函数所写入的外部数据的结果值。返回值就不用说了，被测试函数进行了写操作的参数(输出参数)、成员变量、全局变量、

IO 媒体，它们的预期的结果值都是预期输出。一个测试用例，就是设定输入数据，运行被测试函数，然后判断实际输出是否符合预期。下面举一个与成员变量有关的例子：

产品函数：

```
void CMyClass::Grow(int years)
{
mAge += years;

if(mAge < 10)
mPhase = "儿童";
else if(mAge <20)
mPhase = "少年";
else if(mAge <45)
mPhase = "青年";
else if(mAge <60)
mPhase = "中年";
else
mPhase = "老年";
}
```

测试函数中的一个测试用例：

```
CaseBegin();{
int years = 1;
pObj->mAge = 8;
pObj->Grow(years);
ASSERT( pObj->mAge == 9 );
ASSERT( pObj->mPhase == "儿童" );
}CaseEnd();
```

在输入数据中对被测试类的成员变量 mAge 进行赋值，在预期输

出中断言成员变量的值。现在可以看到老纳所推荐的格式的好处了吧，这种格式可以适应很复杂的测试。在输入数据部分还可以调用其他成员函数，例如：执行被测试函数前可能需要读取文件中的数据保存到成员变量，或需要连接数据库，老纳把这些操作称为初始化操作。例如，上例中 ASSERT(...)之前可以加 pObj->OpenFile();。为了访问私有的成员变量或成员函数，可以将测试类定义为产品类的友元类。例如，定义一个宏：

#define UNIT_TEST(cls) friend class cls##Tester;

然后在产品类声明中加一行代码：UNIT_TEST(ClassName)。

下面谈谈测试用例设计。前面已经说了，测试用例的核心是输入数据。预期输出是依据输入数据和程序功能来确定的，也就是说，对于某一程序，输入数据确定了，预期输出也就可以确定了，至于生成/销毁被测试对象和运行测试的语句，是所有测试用例都大同小异的，因此，我们讨论测试用例时，只讨论输入数据。

前面说过，输入数据包括四类：参数、成员变量、全局变量、IO 媒体，这四类数据中，只要所测试的程序需要执行读操作的，就要设定其初始值，其中，前两类比较常用，后两类较少用。显然，把输入数据的所有可能取值都进行测试，是不可能也是无意义的，我们应该用一定的规则选择有代表性的数据作为输入数据，主要有三种：正常输入，边界输入，非法输入，每种输入还可以分类，也就是平常说的等价类法，每类取一个数据作为输入数据，如果测试通过，可以肯定同类的其他输入也是可以通过的。下面举例说明：

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。

非法输入

非法输入是正常取值范围以外的数据，或使代码不能完成正常功能的输入，如上例中表示年龄的参数，小于 0 或大于 100 都是非法输入，再如一个进行文件操作的函数，非法输入有这么几类：文件不存在；目录不存在；文件正在被其他程序打开；权限错误。

如果函数使用了外部数据，则正常输入是肯定会有的，而边界输入和非法输入不是所有函数都有。一般情况下，即使没有设计文档，考虑以上三种输入也可以找出函数的基本功能点。实际上，单元测试与代码编写是“一体两面”的关系，编码时对上述三种输入都是必须考虑的，否则代码的键壮性就会成问题。

四、 白盒覆盖

上面所说的测试数据都是针对程序的功能来设计的，就是所谓的黑盒测试，另外，单元测试还需要从另一个角度来设计测试数据，即针对程序的逻辑结构来设计测试用例，就是所谓的白盒测试。在老纳看来，如果黑盒测试是足够充分的，那么白盒测试就没有必要，可惜“足够充分”只是一种理想状态，例如：真的是所有功能点都测试了吗？程序的功能点是人为的定义，常常是不全面的；各个输入数据之间，有些组合可能会产生问题，怎样保证这些组合都经过了测试？难于衡量测试的完整性是黑盒测试的主要缺陷，而白盒测试恰恰具有易于衡量测试完整性的优点，两者之间具有极好的互补性，例如：完成功能测试后统计语句覆盖率，如果语句覆盖未完成，很可能是未覆盖的语句所对应的功能点未测试。

白盒测试针对程序的逻辑结构设计测试用例，用逻辑覆盖率来衡量测试的完整性。逻辑单位主要有：语句、分支、条件、条件值、条件值组合，路径。语句覆盖就是覆盖所有的语句，其他类推。另外还有一种判定条件覆盖，其实是分支覆盖与条件覆盖的组合，在此不作讨论。跟条件有关的覆盖就有三种，解释一下：条件覆盖是指覆盖所有的条件表达式，即所有的条件表达式都计算了，不考虑

计算结果；条件值覆盖是指覆盖条件的所有可能取值，即每个条件的取真值和取假值都要计算一次；条件值组合覆盖是指覆盖所有条件取值的所有可能组合。老纳做过一些粗浅的研究，发现与条件直接有关的错误主要是逻辑操作符错误，例如：||写成&&，漏了写！什么的，采用分支覆盖与条件覆盖的组合，基本上可以发现这些错误，另一方面，条件值覆盖与条件值组合覆盖往往需要大量的测试用例，因此，在老纳看来，条件值覆盖和条件值组合覆盖的效费比偏低。老纳认为效费比较高且完整性也足够的测试要求是这样的：完成功能测试，完成语句覆盖、条件覆盖、分支覆盖、路径覆盖。做过单元测试的朋友恐怕会对老纳提出的测试要求给予一个字的评价：晕！或者两个字的评价：狂晕！因为这似乎是不可能的要求，要达到这种测试完整性，其测试成本是不可想象的，不过，出家人不打诳语，老纳之所以提出这种测试要求，是因为利用一些工具，可以在较低的成本下达到这种测试要求，后面将会作进一步介绍。

关于白盒测试用例的设计，程序测试领域的书籍一般都有讲述，普通方法是画出程序的逻辑结构图如程序流程图或控制流图，根据逻辑结构图设计测试用例，这些是纯粹的白盒测试，不是老纳想推荐的方式。老纳所推荐的方法是：先完成黑盒测试，然后统计白盒覆盖率，针对未覆盖的逻辑单位设计测试用例覆盖它，例如，先检查是否有语句未覆盖，有的话设计测试用例覆盖它，然后用同样方法完成条件覆盖、分支覆盖和路径覆盖，这样的话，既检验了黑盒测试的完整性，又避免了重复的工作，达到非常高的测试完整性。不过，这些工作可不是手工能完成的，必须借助于工具，后面会介绍可以完成这些工作的测试工具。

五、 单元测试工具

现在开始介绍单元测试工具，老纳只介绍三种，都是用于 C++语言的。

首先是 CppUnit，这是 C++单元测试工具的鼻祖，免费的开源的单元测试框架。由于已有一众高人写了不少关于 CppUnit 的很好的

文章，老纳就不现丑了，想了解 CppUnit 的朋友，建议读一下 Cpluser 所作的《CppUnit 测试框架入门》，网址是：http://blog.csdn.net/cpluser/archive/2004/09/21/111522.aspx。该文也提供了 CppUnit 的下载地址。

然后介绍 C++Test，这是 Parasoft 公司的产品。［C++Test 是一个功能强大的自动化 C/C++单元级测试工具，可以自动测试任何 C/C++函数、类，自动生成测试用例、测试驱动函数或桩函数，在自动化的环境下极其容易快速的将单元级的测试覆盖率达到 100%］。

［］内的文字引自：http://www.superst.com.cn/softwares_testing_c_cpptest.htm，这是华唐公司的网页。老纳想写些介绍 C++Test 的文字，但发现无法超越华唐公司的网页上的介绍，所以也就省点事了，想了解 C++Test 的朋友，建议访问该公司的网站。华唐公司代理 C++Test，想要购买或索取报价、试用版都可以找他们。老纳帮华唐公司做广告，不知道会不会得点什么好处？

最后介绍 Visual Unit，简称 VU，这是国产的单元测试工具，据说申请了多项专利，拥有一批创新的技术，不过老纳只关心是不是有用和好用。［自动生成测试代码 快速建立功能测试用例 程序行为一目了然 极高的测试完整性 高效完成白盒覆盖 快速排错高效调试 详尽的测试报告］。

［］内的文字是 VU 开发商的网页上摘录的，网址是：http://www.unitware.cn。前面所述测试要求：完成功能测试，完成语句覆盖、条件覆盖、分支覆盖、路径覆盖，用 VU 可以轻松实现，还有一点值得一提：使用 VU 还能提高编码的效率，总体来说，在完成单元测试的同时，编码调试的时间可能还会缩短。算了，不想再讲了，老纳显摆理论、介绍经验还是有兴趣的，因为可以满足老纳好为人师的虚荣心，但介绍工具就觉得索然无味了，毕竟工具好不好用，合不合用，要试过才知道，还是自己去开发商的网站看吧，可以下载演示版，还有演示课件，老纳念经去也，阿弥陀佛。

使用 TCL 脚本读取配置文件

作者：叶晖 兰海

摘　要：unix 下使用 TCL 脚本读取配置文件；错误处理.

关键词：TCL、配置文件、unix

一、 应用范围

在实际工作中，TCL 脚本对于一些简单的工作裨益甚大。通常编写脚本都有一定的模式，首先从配置文件读入配置项，初始化变量，然后进行处理，最后输出，在程序的运行过程中需要把一些信息写入日志文件，而调试信息写入日志文件和直接输出到屏幕都可以。

使用任何一种脚本，通常也都会根据实际情况建立起自己的函数库，而这个函数库中对配置文件配置项的读取无疑是非常基本而重要的。

这篇文章的本意是引导刚接触 TCL 脚本的朋友尽快上手，所以有些细节的说明文字比较细。

二、 程序讲解

构建一个配置文件，尽可能的接近实际应用：

文件名：config.ini

文件内容：

```
#
# 这是注释行
#
[]
key1=value1              # 注释
```

[section1]

key2=value2

[section2]

key1=value2

[section1]

key1=xxxxxx # 这是最后的返回

[section1]

key1=value2

现在开始对程序的说明：

;#----------

;# 功能：从配置文件中读取配置项

;# 输入：1. configFile ：配置文件名称

;# 2. Section：段名称

;# 3. Key：关键字

;# 4. Comment：注释符，缺省为井号

;# 5. Equal：关键字和值的分隔符，缺省为等号

;# 输出：1. Value：相应的值

;#----------

1. 过程的定义：

格式：proc name args body

要点：

Ø 参数列表使用花括号引起；

Ø 变量没有类型；

Ø 变量之间使用空格间隔；

Ø 如果参数有缺省值，使用花括号引起，并赋值

```
proc getConfig { configFile Section Key {Comment "#"} {Equal "="}}
{
        set Value ""                          ;# 记录过程返回的值
        set FindSection 0                     ;# 记录是否找到了
section
```

2. 错误的处理

格式：catch script ?varName?

功能：执行 script，如果成功返回 TCL_OK(0)，否则返回 TCL_ERROR(1)，提示结果存在 varName 中。比如下面如果打开文件成功，errMsg 返回的就是类似 file3 这样的字符串，此时 err 返回

的是 TCL_OK，就是零；如果文件不存在，errMsg 返回的类似：couldn't open "config1.ini": no such file or directory，此时 err 返回 TCL_ERROR，就是一。

3. 文件的读写

格式：openfile fileName ? access ? permission

讲解：

Ø fileName 是文件名称；

Ø access 是存取模式，可以为 r, r+, w, w+, a, a+ 六种模式，r 、r+ 和 a 三种模式文件必须已经存在，其他三种模式文件不存在就创建一个。本文用的 r 模式，所以文件不存在会提示错误，而不是自动建立一个；

Ø permission 是权限，举例说明：
有权限为：000 000 000 ：第一组三位为 user 权限；第二组三位为同组其他用户的权限；第三组三位为其他组所有人的权限。每个三位的权限依次代表读，写，执行。如果有相应的权限就设置为一，没有设置为 0。然后三位为组转成十进制数。

Ø 文件打开后就可以使用其文件 id，使用完后记得关闭文件

```
;# 打开配置文件

set err [catch {set fileid [open $configFile r]} errMsg]

if {$err == 1} {

        puts "errMsg : $errMsg"

        return $Value

}


;# 成功打开文件后，一行一行的加以分析

set rowid 0                                    ;#记录当前行数，
程序调试时打印调试信息使用的

seek $fileid 0 start                           ;# 定位
到文件头

while {[eof $fileid] != 1} {                   ;# 读取
文件内容
```

4. 变量的自动增长使用命令 incr，也可以使用 set rowid [expr $rowid + 1]，显然前者更简捷

```
                incr rowid                                ;#
```

记录行数，从一开始

```
                ;# 读出一行

                gets $fileid line
```

5. 先期处理行，因为注释有两种情况，行中的注释和整行注释，先去掉注释，然后去左右空格就可以得到真正需要的内容；如果先去空格，再去注释，由于行中的注释和内容之间有空格，这样最后得到的内容在去掉行中注释后，会在后面留下一些空格。所以需要先去掉注释，后去掉空格。

6. 得到一个字符串在另一个字符串中首先出现的位置，使用函数 string first

   格式：string first string1 string2

   讲解：返回 string1 在 string2 中第一次出现的位置；如果 string1 不在 string2 中，返回-1

7. 下面有一个细节是初学者经常犯错的地方，那就是：} else { ，这里必须严格的 花括号，空格，else，空格，花括号，不能把花括号写到上一行或者下一行。

8. 返回一个字符串的子串使用 string range 函数

   格式：string range string1 frompos topos

讲解：取 string1 的从 frompos 到 topos 之间的字符串，注意这里字符串下标是从零开始的，所以用 string length string1 得到的字符串的长度比字符串最后一个字符的位置值要大一。如果取一个字符串中的某个字符就可以使用函数 string index string1 pos。

9. expr 和 unix shell 中一样，是进行数学运算的函数。

```
;# 先去掉注释，再去掉两端的空格

set commentpos [string first $Comment $line]        ;# 得到注释符号的位置

if { $commentpos == 0 } {

;# 行以注释符号开头，忽略掉该行

} else {

if { $commentpos != -1 } {          ;# 行中有注释符号,去掉注释

set line [string range $line 0 [expr $commentpos-1]]

}
```

```
                    set line [string trim
$line]          ;# 去掉两端的空格

                    ;# puts "$rowid : line : $line"
```

10. 在 tcl 脚本中，循环中有 break 命令跳出循环，continue 跳到循环的开头。不过因为括号的使用，其实这里写不写 continue 都没有问题。

```
                    ;# 如果是空就继续循环

                    if { $line == "" } {

                        ;# 回循环

                        continue

                    } else {

                        ;# 先找 section

                        set linelen [string length
$line]          ;#字符串长度

                        set lastpos [expr $linelen -
```

```
1 ]                ;#字符串最后的位置

                                        ;# puts "$rowid :len: $linelen
lastpos: $lastpos"

                                        if { [string index $line 0] ==
"\[" && [string index $line $lastpos] == "\]" } {

                                                ;# 如果是查找的
section，修改标志位；如果不是相应的 section，需要将标志重新赋
值

                                                if { [string range
$line 1 [expr $lastpos - 1 ]] == $Section } {

                                                        ;#      puts
"$rowid: find section : $Section"

                                                        set
FindSection 1

                                                } else {

                                                        set
FindSection 0

                                                }

                                        } else {
```

```
                                        ;#  已 经 找 到 了
section 才继续找 key

                                        if { $FindSection ==
1 } {

                                            set
equalpos [string first $Equal $line]    ;# 得到等号的位置

                                            if
{ $equalpos != -1} {

                                                ;#
如果就是找寻的 key,结束循环

                                                if
{ [string range $line 0 [expr $equalpos - 1]] == $Key } {


puts "$rowid: find key"


set Value [string range $line [expr $equalpos + 1] [string length $line]]


puts "$rowid: find value: $Value"
```

```
break

                                                                    }

                                                                } else {

                                                                    ;#
回循环

                                                                }

                                                            } else {

                                                                ;#  回循环

                                                            }

                                                        }

                                                    }

                                                }

                                            }


                                            ;#  关闭文件

                                            close $fileid
```

```
        return $Value

}
set val ""

;# 测试正常情况下

set val [getConfig "config.ini" "section1" "key1"]

puts "val : $val"

;# 测试文件不存在的情况下

set val [getConfig "config1.ini" "section1" "key1"]

puts "val : $val"
```

该程序在 unix 环境下调试通过。

三、 伪代码

为了便于理解程序，特写了下面的伪代码：

```
打开配置文件

        IF 出错 THEN

            过程结束
```

END IF

文件打开成功，定位到文件头

WHILE 没有到文件尾

读出一行

用 WinRunner 实现软件的全球化测试

作者: 月白

摘　要：本文采用循序渐进的方法详细的介绍了如何用 WinRunner 实现软件的全球化测试。当然，单靠 WinRunner 本身是无法完全实现的，我们开发了一个小程序 COFAL 来帮助 WinRunner 实现全球化测试。通过学习这篇文章，您可以掌握：

- WinRunner 的在全球化测试中的缺陷
- WinRunner 本身可用于全球化测试的地方
- COFAL 如何帮助 WinRunner 实现全球化测试
- COFAL 的实现细节

关键字：Globalization（g11n），Internationalization（i18n），localization（l10n），Code Once Fit All Language（COFAL）

1. 背景

全球化已经成为当今软件发展的趋势，许多大型跨国软件公司都在亚洲设立了自己的专门从事全球化测试的部门。2004 年的 8 月，我加入 Oracle 甲骨文北京研发中心，正式成为这其中的一员，我测试的软件是 Oracle Application Server 10g，以下简称 Oracle AS。Oracle AS 是一个基于 J2EE 架构的应用系统，详细的介绍您可以参考 OTN 上的相关文档。

1.1. 全球化中的概念

全球化的英文是 Globalization,由于单词较长，所以为了书写方便，通常缩写为 G11N，中间的 11 代表首字母”G”和尾字母”N”之间省略的 11 个字母。

引用”中国本地化”网站上对全球化的定义：Globalization 是

使产品或软件进入全球市场而进行的有关的商务活动。包括正确的国际化设计，本地化集成，以及在全球市场进行的市场推广、销售和支持的全部过程。

全球化中与我们测试直接相关的有国际化设计和本地化集成。

国际化的英文是 internationlization，由于单词较长，通常缩写为 I18N，中间的 18 代表首字母”I”和尾字母”N”之间省略的 18 个字母。引用”中国本地化”网站上的定义：国际化设计是指设计一个适用于多种语言和地区的应用程序的过程。适用于多种语言和地区的含义是当使用不同语言及处于不同的地区的用户在使用这个应用程序时，应用程序必须使用他们能看懂的语言和符合他们文化习惯来显示信息。

本地化的英文是 localization,由于单词较长，通常缩写为 L10N，中间的 10 代表首字母”L”和尾字母”N”之间生罗的 10 个字母。引用”中国本地化”网站上的定义：本地化是指将产品或软件针对特定国际语言和文化进行加工，使之符合特定区域市场的过程。真正的本地化要考虑目标区域市场的语言、文化、习俗和特性。通常包括改变软件的书写系统（输入法）、键盘使用、字体、日期、时间和货币格式等。

Locale 表示表示一个特定的地理、政治或文化的区域，在 java 中有 Locale 类，我们会在 1.3 小节中给出详细的描述

1.2. 全球化测试的内容

简单的说，全球化测试主要是测试软件的处理数据和显示数据的功能。以 Oracle AS 为例：

- 处理不同的字符集（encoding）数据

Oracle Internet Directory（简称 OID）是一个 LDAP 服务器，数据保存在 Oracle 数据库中，现在想测试它创建用户的功能，要求用户的 DN 可以为不同国家的字符集，通俗的说，可以创建英文的 DN，简体中文的 DN 和日文的 DN 等。当然具体可以创建哪些字符集的

DN 也要看当前 Oracle 数据库的字符集，只是那些在可以和当前字符集正确转换的字符集中的 DN 才可以正确的创建，否则很有可能无法创建或者创建的结果错误，如我们经常会看到的一些数据变成了问号（？）。

● 动态显示与 Locale 有关的数据

Oracle Delegated Administration Services（简称 DAS）是一个通过 web 页面访问的组件，页面的编码方式为 UTF8，要求当选择不同的浏览器语言时，以下各项都可以显示为与当前 Locale 相符的形式：

如某个页的标题，在英文下为“Home”，在中文下为“主页”；某个按钮上的标签，在英文下为“OK”，在中文下为“确定”。

☆ 表示日期、时间、时区和货币等的文字

如某个页上的一段表示出生日期的文字，在英文下显示为“January 1, 1976”，在中文下显示为“1976 年 1 月 1 日”。

1.3. Java 程序的国际化设计

Java 语言是平台无关的，它采用双字节字符编码（UTF16），在解决国际化问题上有天生的优势。下面我要介绍的是 Java 中“动态显示与 Locale 有关的数据”的原理。

这里要用到的几个主要类都在 java.util 包（package）中，包括有 Locale 、 ResourceBundle 、 ListResourceBundle 、 PropertyResourceBundle 等，其继承关系如下图所示：

● Locale

该类包含对主要地理区域的地域化特征的封装。通过设定 Locale，我们可以为特定的国家或地区提供符合当地文化习惯的字体、符号、图标和表达格式。例如，我们可以通过获得特定 Locale 下的 Calendar 类的实例，显示符合特定表达格式的日期。Locale 有

以下三个构造函数：

◆ Locale(String language)

◆ Locale(String language,String country)

◆ Locale(String language,String country,String variant)

language 参数：代表两个小写英文字符的 ISO 语言编码，如 zh 表示 Chinese,可用的语言编码可以参考：

http://www.ics.uci.edu/pub/ietf/http/related/iso639.txt

country 参数：代表两个大写英文字符的 ISO 国家或地区编码，如，CN 表示 China，TW 表示 TAIWAN，国家代码对照表如下：

http://userpage.chemie.fu-berlin.de/diverse/doc/ISO_3166.html

variant 参数：代表与供应商或浏览器相关的代码。如，WIN 表示 windows，MAC 表示 Macintosh，POSIX 表示 POSIX。当有两个 variant 存在的话，用下划线（uderscore）连接，并把最重要的 variant 放在前面。

下面是几个典型的 Locale 的例子

Locale("ja")

Locale("zh","CN")

Locale("zh","TW","WIN")

Locale("es","ES""Traditional_WIN")

Locale.getDefault()，得到当前 Java 虚拟机的宿主系统上默认的 Locale

● ResourceBundle

该类是一个抽象类，它定义了三个静态方法来获得具体的实现类（ListResourceBundle 的子类或 PropertyResourceBundle 类）的实例：

☆ static final ResourceBundle getBundle（String baseName）

等同于调用：

getBundle(baseName,Locale.getDefault(),this.getClass.getClassL

oader())

使用的是系统缺省的 Locale。

☆ static final ResourceBundle getBundle（String baseName,Locale locale）

等同于调用：

getBundle(baseName,locale,this.getClass.getClassLoader())

使用的是参数 locale 指定的 Locale。

☆ static final ResourceBundle getBundle（String baseName,Locale locale,ClassLoader loader）

下面我们来说说 baseName 参数和 locale 参数。

BaseName 参数指定的是一组 ReourceBundle 的公共的基础名称，例如，设 baseName 等于“TestBundle”。如果用 ListResourceBundle 子类来实现，则要有如下这样的类:TestBundle.class、TestBundle_zh_CN.class 和 TestBundle_fr.class 等；如果用 PropertyResourceBundle 来实现，则要有如下这样的属性文件：TestBundle.properties、TestBundle_zh_CN.properties 和 TestBundle_fr.properties 等。

locale 参数和选择策略一起决定运行时具体选择这组 ResourceBundle 中的哪一个。

假设 locale 参数指定的 Locale 为（language1,country1,variant1），系统默认的 Locale 为（language2,country2,variant2），则按照以下优先级的顺序查找最满足条件的 ResourceBundle：

- baseName + "_" + language1 + "_" + country1 + "_" + variant1
- baseName + "_" + language1 + "_" + country1
- baseName + "_" + language1
- baseName + "_" + language2 + "_" + country2 + "_" + variant2

- baseName + "_" + language2 + "_" + country2
- baseName + "_" + language2
- baseName

在每一种情况下，会先尝试按 ListResourceBundle 类的方式加载，失败后会再尝试按照访问属性文件的方式加载 PropertyResourceBundle 类。如果所有这些情况都没有找到的话最后会抛出一个 MissingResourceException 的异常。

注意，在第一个 getBundle 静态函数中 locale 参数指定的 Locale 就是系统默认的 Locale。

- ListResourceBundle

该类继承 ResourceBundle 类，也是一个抽象类。它实现了 ResourceBundle 类中的抽象函数 getKeys()和 handleGetObject(String key)，并提供了一个抽象函数 getContents()。在应用中，通过创建继承 ListResourceBundle 的子类来实现 ResourceBundle。要求子类必须实现 getContents 函数并提供一个包含有一组属性对的数组，如：

```
package oracle.cdc.sgt.unicode;
import java.util.ListResourceBundle;
public class MResources extends ListResourceBundle {
    public Object[][] getContents() {
       return contents;
    }
    static final Object[][] contents = {
       {"s1", "Home"}
    };
 }
package oracle.cdc.sgt.unicode;
import java.util.ListResourceBundle;
public class MResources_zh_CN extends ListResourceBundle {
    public Object[][] getContents() {
```

```
        return contents;
    }
    static final Object[][] contents = {
        {"s1", "主页"}
    };
}
```

下面是一个 java 类根据不同的 Locale 从相应的 ListResourceBundle 子类中取数据来显示：

```
package oracle.cdc.sgt.unicode;
import java.util.ResourceBundle;
import java.util.Locale;
public class TestListResourceBundle {
    public static void main(String[] args) {
      ResourceBundle messages;
      Locale curloc;
      try {
        if (args.length != 2) {
          curloc = Locale.getDefault();
        } else {
          curloc = new Locale(args[0], args[1]);
        }
        messages = ResourceBundle.getBundle(
            "oracle.cdc.sgt.unicode.MResources", curloc);
System.out.println(messages.getString("s1"));
      } catch (Exception e) {
        System.out.println(e);
      }
    }
```

```
}
```

把这三个类加入到 classpath 中，运行“java TestListResourceBundle zh CN”或在简体中文操作系统上运行“java TestListResourceBundle”，打印出“主页”；运行“java TestListResourceBundle en”或在英文操作系统上运行“java TestListResourceBundle”，打印出“Home”。

- PropertyResourceBundle

继承 ResourceBundle 类，它不是抽象类，也不需要创建它的子类。与 ListResourceBundle 相同的是它也实现了 ResourceBundle 类的抽象函数 getKeys()和 handleGetObject(String key)；不同的是，它是从属性文件（.properties）中读入属性对的。例如，

定义如下一组 properties 文件，并加入到 classpath 中：

MResources.properties：

```
s1=Home
```

MResources_zh_CN.properties

```
s1=主页
```

下面是一个 java 类根据不同的 locale 从相应的 Properties 文件中取数据来显示：

```
package oracle.cdc.sgt.unicode;
import java.util.Locale;
import java.util.ResourceBundle;
public class TestPropertyResourceBundle {
  public static void main(String[] args) {
    ResourceBundle messages;
    Locale curloc;
    if (args.length != 2) {
        curloc=Locale.getDefault();
    } else {
```

```
            curloc = new Locale(args[0],args[1]);
        }
        messages = ResourceBundle.getBundle(
    "oracle/cdc/sgt/unicode/MResources",
     curloc);
        System.out.println(messages.getString("welcome"));
      }
    }
```

运行的方式和结果同 TestListResourceBundle 一样。

留意一下 TestListResourceBunlde 和 TestPropertyResourceBundle 唯一不同的地方就是在调用 getBundle 函数的那个语句，按照我们上面所说的，完全可以统一写成：“oracle.cdc.sgt.unicode.MResources”，因为 getBundle 会缺省先找基础名称为 MRseources 的类，失败后再找基础名称为 MResources 的属性文件，在查找属性文件前它会自动把“.”转换为“/”。当然，如果通过存在基础名称为 MResources 的类和属性文件时，也可以通过直接使用“oracle/cdc.sgt/unicode/MResources”来略过查找基础名称为 MResources 的类。

当然，java 程序的国际化设计并不只是这么简单，当涉及日期和时间显示等问题时，还可以利用 java.text 包以及 java.util 包中的 TimeZone、SimpleTimeZone 和 Calendar 等类进行辅助处理。我们就不在这里详细叙述了，您只需要记住一个 ResourceBundle 的概念就可以了，本文的后续部分都是围绕着这个概念展开的。

2. WinRunner 调研

WinRunner 适合于测试那些有图形操作界面的组件。目前，我们手头可用的版本 WinRunner7.5，启用 Web 和 Java 插件（plugin）。

让我们先从 WinRunner 的技术特点说起吧。

2.1 WinRunner 的技术特征

由于本文不是专门介绍 WinRunner 的，所以只列举一些 WinRunner 的重要特征。注意：这里定义了一些非官方的术语，为的是便于您的理解。

WinRunner 将对象（object）分为两种：窗口（window）和子对象，任一个子对象都隶属于一个窗口。

注意：窗口也是对象，如一个页面就是一个窗口。

- WinRunner 通过一组属性来唯一的识别窗口，也就是说不能有所有属性值都相同的多个窗口；同样的，WinRunner 通过一组属性来唯一的识别同一个窗口下的子对象，也就是说，在同一个窗口下，不能有所有属性值都相同的多个子对象。
- 如果把对象的所有属性的集合称为对象的定义，WinRunner 可以把对象的定义保存在以下两个地方：

☆ 独立 script 的单独的扩展名为 GUI 的文件

简称为 GUI 文件，同时为每个 object 定义了一个绝对逻辑名，有了绝对逻辑名就一定有相对逻辑名。对于窗口来说，它的绝对逻辑名等于它的相对逻辑名；对于子对象，它的绝对逻辑名等于它隶属的窗口的绝对逻辑名后面加一个”.”再加上它的相对逻辑名。在 script 开始部分导入 GUI 文件，在后面部分中只需要写出对象的绝对逻辑名，就可以从 GUI 文件中获得这个对象的定义了。如：

```
"OracleAS Certificate Authority-Certificate Management":
{
 class: window,
 MSW_class: html_frame,
 html_name: "!OracleAS Certificate Authority-Certificate Mana.*"
}
{
 ltree_state: open,
```

list_open_data: close

}

"OracleAS Certificate Authority-Certificate Management"."Advanced Search":

{

class: object,

MSW_class: html_text_link,

html_name: "Advanced Search"

}

"OracleAS Certificate Authority-Edit Policy Result: UniqueCertificateCo":

{

class: window,

MSW_class: html_frame,

html_name: "!OracleAS Certificate Authority-Edit Policy Resu.*"

}

{

rtree_state: open,

ltree_state: open,

list_open_data: close

}

☆ script 本身

把 object 定义写在 script 是可以的。一种方法是象 GUI 文件那样在 script 的开始部分为对象定义一个绝对逻辑名，这样在 script 的后续部分就可以通过这个绝对逻辑名来访问对象的；另一种方法是不为对象定义绝对逻辑名，而是在每个要访问对象的地方，直接写该对象的定义。这两种在 script 中定义对象的方法我们都不推荐，

第一种完全就是 GUI 文件在 script 中的实现，那么为什么不它放在 GUI 文件中统一管理呢，第二种虽然省略了导入 GUI 文件的一步，但是维护起来更麻烦了，如果对象的属性发生变化，那就要修改所有脚本中所有这样定义了该对象的地方。

也许我们还没有意识到在 script 中定义对象的好处，但是存在就是道理。如：

```
#Gui Objects initialization
set_window("{ class: window, MSW_class: html_frame, html_name: \"OracleAS Wireless Tools\"}",151);
list_select_item("{ class: list, MSW_class: html_combobox, html_name: matchType}","Matches");
rc=global_web_obj_text_exists(access_info,"Wireless and Voice Settings");
if(text!="Application Links")
set_window("{ class: window, MSW_class: html_frame, html_name: \"Oracle Enterprise Manager - Notification Event Collector: "¬ification&"\"}",8);
rc=global_web_obj_text_exists(text_object,"#1","#1","Access Information","","");
set_window("{ class: window, MSW_class: html_frame, html_name: \"OracleAS Wireless Tools\"}",151);
```

3.1 WinRunner 在全球化测试中的局限性

在 1.1“全球化测试的内容”一节中我们知道，要在不同的 Locale 下测试软件的处理数据和显示数据的能力。在不同 Locale 下，WinRunner 赖以识别对象的属性列中有的属性也可能不同，因此在不同的 Locale 下，同一个对象的定义也可能不同。如同一个窗口，在英文下的 html_name 属性值为“OracleAS Certificate Authority-Certificate Management”，在简体中文下的 html_name 属性

值为“OracleAS Certificate Authority-证书管理”。也就是如果用该窗口在英文下的定义是无法在简体中文或其他 Locale 下识别该窗口的。

换句话说，在一种 Locale 下录制的脚本，不论对象定义是保存在 GUI 文件中还是保存在 script 中，都无法直接拿到另一种 Locale 下直接运行。注意：所谓直接拿到，也包括进行少量的修改。

虽然可以在不同的 Locale 下用 WinRunner 录制各自的脚本，但是这并不是我们所希望的，那样做的成本是非常高的。

我们的目标是只在一种 Locale 下录制脚本，经过一定处理后，就可以在其他 Locale 下使用，即 Code Once Fit All Language(简称 COFAL)。

3.1 WinRunner 满足 COFAL 的技术可行性

既然 WinRunner 是通过对象的定义(一组属性)来标识对象的，那么我们就要研究对象的属性在不同的 Locale 下有什么不同：

- 有的对象在不同的 Locale 下所有属性值都不变
- 有的对象在不同的 Locale 下部分属性发生变化

只要找到变化的属性的规律和属性值的来源，并用自动化的方法来修改这些属性，就可以基本上满足只录制一次的需求。

1、 如果对象的定义保存在 GUI 文件中

假如在英文下录制了一套脚本，该套脚本公用一个 GUI 文件 global.gui，我们要找到一个自动化的方法，生成该 GUI 文件在其他 Locale 下对应的 GUI 文件，如 global_zh_CN.gui 和 global_fr.gui 等。这样，在不同 Locale 下，通过使用不同的 GUI 文件就可以用同一套脚本运行了。这样看来虽然有多个 GUI 文件，但是脚本只有一套，其他的 GUI 文件又是自动生成的，基本上满足了 COFAL 的要求。

2、 如果对象的定义保存在 script 中

假如在英文下录制了一套脚本，脚本都保存在 tina 目录下，对象的定义都保存在 script 中。我们要找到一个自动化的方法，转化

该 script 中对象的定义到不同的 Locale 下的定义，并把转化的结果保存在新的目录下，如 tina_zh_CN 和 tina_fr 目录等。这样，在不同的 Locale 下，通过使用不同目录下的脚本就可以了。这样看来虽然有多套脚本，但是只录制了一次，其他的都是自动生成的，也基本上满足了 COFAL 的要求。

下面我们会把转化对象定义统一称为“翻译”，接下来要介绍的就是以 COFAL 命名的一个实现自动化翻译的小工具。

3、 3. 自动翻译工具 COFAL 简介

CORAL 是 Code Once Fit All Language 的缩写，它是专门为配合 WinRunner 的全球化测试而开发一个工具，code once fit all language 的意思是只需要在一种语言下编写脚本，就可以在所有语言下运行。用 COFAL 来实现 script 和 GUI 文件的自动翻译。

3.1 技术原理

3.1.1 Java 应用程序级数据翻译

Oracle AS 是一个基于 J2EE 架构的应用程序，它是通过我们在 1.2 节中介绍的 java 数据绑定机制来实现国际化的，也就是说那些需要翻译的属性值其实都是保存在 ResourceBundle 中。除了前面说过的 ListResourceBundle 和 PropertyResourceBundle 外，Oracle AS 还把部分 ResourceBundle 保存在数据库的表中，在运行时根据不同的语言环境用绑定的 key 动态的从表中检索出对应的值。

由此可知，程序本身是通过关键字(key)结合指定的 locale 来获得值(value)的；而现在是想在已知值(value)和指定的 locale 的情况下，获得该 key 在其他 locale 下的值(value)，这是一个逆向的过程。

为了描述方便，我们用 key 代表关键字，用 prevalue 代表当前可得的值，用 postvalue 代表翻译后的值。

自动翻译的原理是：

a) 事先定位好 resourcebundle 的保存位置。

b) 翻译时，从 script 或 GUI 文件中提取出 prevalue，然后用 prevalue 在 ResourceBundle 中查询出 key，再用 key 从 ResourceBundle 中查询出 postvalue。

c) 用 postvalue 替换 prevlaue，把替换的结果保存成新的语言版本。

在 COFAL 中，我们把 Oracle AS 用到的 ResourceBundle 统一保存在一张 Oracle 数据库表 GLOBALRES 中。翻译时通过 JDBC，用 prevalue 从表中 select 出 key，再用 key 去 select 出 postvalue。

GLOBALRES 表的结构如下：

列	类型	描述
Component	varchar2(60)	Key 所属的组件
Version	varchar2(60)	组件的版本
Prolang	varchar2(60)	从 ResourceBundle 文件名中提取出来的表示语言的串，如从 TestBundle_zh_CN.properties 中提取出的 zh_CN
Key	varchar2(400)	从 ResourceBundle 文件内容中提取出来的（key-value）对中的 key
value	nvarchar2(400)	从 ResourceBundle 文件内容中提取出来的（key-value），因为有的 value 非常长，已经超出了 4000 个字节的字段长度限制，所以如果 value 超过 4000 个

		字节，提取 vlaue 中前 400 个字节否则保留全部字节。注意是 400 个字节不是 4000 个字节，因为在 WinRunner 中完全可以通过正则表达式的使用将字符串截短，不用保留那么长的串。
Value1	clob	这是一个备份字段，当 value 超过 4000 个字节时把整个 value 备份到这里，否则为空
location	varchar2(1200)	保留（key-value）对的出处，即它来自于哪个 ResourceBundle

注意：component 和 version 字段是为了能更有效的翻译而引入的，在导入 resourcebundle 时应该确定它属于哪个 component 的哪个 version，这样在翻译时就可以通过增加 where 判断条件来更具体化一个查询。

抛开如何获得 prevalue 先不说，最终要进行的 select 查询就是以下两种

● SQL 精确查询

☆ 获得 key 的 SQL 语句

select key from GLOBALRES where value=’Add Another Row’ and prolang=’en’ and component=’UIX’ and version=’10.1.2.0.2’

☆ 获得 postvalue 的 SQL 语句

假设上面的 SQL 语句查询出来的 key 是 TABLE_ADD_ROW_SINGLE_TEXT，则获得 postvalue 的语句为：

select value from GLOBALRES where key= ’TABLE_ADD_ROW_SINGLE_TEXT’ and prolang=’zh_cn’ and component=’UIX’ and version=’10.1.2.0.2’

● SQL 模糊查询

☆ 获得 key 的 SQL 语句

select key from GLOBALRES where value like 'Certificate

Mana%') and prolang=’en’ and component=’oca’ and version=’10.1.2.0.2’

☆ 获得 postvalue 的 SQL 语句

假设上面的 SQL 语句查询出来的 key 是 OCAUIAdminTabText，则获得 postvalue 的语句为：

select value from GLOBALRES where key= ’OCAUIAdminTabText’ and prolang=’zh_cn’ and component=’oca’ and version=’10.1.2.0.2’

看到这里，您也许会有一个疑问：从 key 到 postvalue 是一对一的关系，但是从 prevalue 到 key 则是多对一的关系，比如可能有多个 key 在英文下的 value 都是 OK,这就是所谓的对象多翻译的情况，我们将在 3.2.4 小节说明如何处理这种情况。

3.2.1 操作系统级数据翻译

有时测试要打开一些操作系统级的窗口使用里面的一些对象，由于我们不知道如何获得它们的 ResourceBundle,所以只好用一种最笨的办法：

（1） 第一次将用到的操作系统级的对象手动搜集（可以使用 WinRunner 的 Spy 来录制）在一起，保存成在一个单独的 resourcebundle 中；

（2） 以后再用到时，从这个 resourcebundle 里找。

在 COFAL 中，我们单独定义了一张表 SYSRES 来保存操作系统级的 resourcebundle，表的结构如下：

字段名	字段类型	描述
win	varchar2(120)	窗口的绝对逻辑名

obj	varchar2(120)	对象的相对逻辑名，当对象是窗口是该字段那为 null
value	nvarchar2(240)	对象的需要翻译的属性值
lang	varchar2(10)	语言

这张表的结果完全是按照 WinRunner 中“窗口和子对象”的模式定义的，我们规定：

- win 字段和 obj 字段都只能是英文字符串，取对象在英文环境下的名字。如“Security Alert”窗口中的“OK”按钮对应的 win 字段为“Security Alert”，ob 字段为“OK”。
- value 字段实际上就是对象在不同语言环境下的名字。如“Security Alert”窗口在英文下的 value 是“Security Alert”，在简体中文下的 value 是“安全警报”。

由于我们作出了这样的规定，操作系统级数据只能从英文翻译到其他语言，且 prevalue 等于 key，这样就可以用 key 直接查找 postvalue，省略了从 prevalue 到 key 的过程。

另外，在 SYSRES 中没有象 GLOBALRES 中的 component 和 version 字段，这是因为操作系统的版本比较固定，不象产品那样会有很多个 release，同时获得 component 对我们来说意义也不大。当然，您也可以根据您的实际情况添加这两个或其他字段。

3.2 翻译策略

需要翻译的 prevalue 来自于对象的属性值，但并不是所有的对象都需要翻译，即使需要翻译也不是所有的属性都需要翻译，即使找到了需要翻译的属性，它的值也一定都可以直接翻译，可能要经过一些特殊处理才能够翻译。下面我们就来一一说明。

3.2.1 通过 class 属性来判断 object 是否需要翻译

每个 object 的定义里都肯定要包含它的 class 属性，通过 class 属性可以判断出该 object 是否需要翻译。

3.2.1.2 不需要翻译的 class

以下 class 类型一般不需要翻译：

- edit
- combobox
- check_button
- list
- radio_button

3.2.1.2 需要翻译的 object

除了以上那些不需要翻译的 class 外，其他的 class 类型基本上都需要翻译。

注意，以上我们所说的 class 都是 WinRunner 能够理解的标准的 web class，如果您自定义了一些新的 class 类型，是否需要翻译要由实际情况而定。

在 Oracle AS 中我们没有定义新的 class 类，一些非标准的 object 也可以用标准的 class 类型定义，如 Oracle AS 中的某一按钮的定义如下：

```
"OracleAS Certificate Authority-Advanced Search".Go:
{
 MSW_class: html_rect,
 class: object,
 html_name: "Go",
 location: 0
}
```

建议，尽量用 WinRunner Spy 来录制应用程序中的 object，不要

一开始就自己手动定义。这样你才能发现 WinRunner 是如何识别你的 object 的。

3.2.1.3 需要翻译的属性

一般情况下，对于那些需要翻译的 object，下列属性需要翻译：

- label
- html_name
- attached_text

注意：一个 object 一般只具有上列三个属性中的一个，有哪个就翻译哪个。另外，需要翻译的属性可能不只是这三个，但是在我们的测试中这三个属性就足够了。

3.2.3 判断是 Java 应用程序级对象还是操作系统级对象

由 3.1.1 和 3.1.2 两个小节可以看出，Java 应用程序级对象和操作系统级对象的处理是不一样的。如果一个 GUI 文件或 script 文件中同时包含这两种类型的对象定义时，该怎么办呢？由于操作系统级对象的数量比较少，所以每次我们尝试把对象作为操作系统级对象进行翻译，如果不成功作为 Java 应用程序级对象进行翻译，也就是说先 SYSRES 中查找，不成功再在 GLOBALRES 中查找。

3.2.3 翻译规则及其优先级

把从 GUI 文件中提取出的属性值看作是 prevalue，配合其他查询条件最好的情况是可以找到 value 字段值就等于 prevalue 的行，进而取出 key 字段。但有时没有这么幸运，那么是否就要开始模糊查询了，要知道对 value 字段使用 like 查询可能会找到很多行，也就是说很多 key。无论如何这应该是最后一招，在这之前我们还应该再做些什么的。在实践中我们发现，WinRunner 在学习对象的时候可能会对得到的属性值做一定处理，而且 java 程序在绑定数据时也可能对绑定的 key 返回的 value 一定的处理。而我们只要找到这些规则，然后通过反向运用，就可以得到一些更纯粹的子 prevlaue，翻译这些子 prevalue 再把翻译的结果按规则正向组合起来作为最终的

postvalue。

下面要介绍的这些语法规则是在我们的测试中用到的。

3.2.3.1 WinRunner 的语法规则

WinRunner 有许多语法规则，我们用到的就是“!.*”。

使用 spy 的朋友都知道，如果串过长，WinRunner 会自动把它截短并在串的头和尾分别加上“!”和“.*”。其中：“!”表示省略，必须放在串首；“.*”表示任意字符串，可以放在除串首外任何位置，一般放在串尾。

3.2.3.2 SQL 语法规则

在 SQL 语句中，用单引号作为字符串的边界符，所以不论是做 select 精确查询还是 select 模糊查询，都要把 prevalue 中的单引号 escape 掉，escape 符号是 SQL 中的 escape 符：“’”(单引号)。

注意：java 中字符串的边界符是双引号，escape 符号是：“\”(反斜杠)

3.2.3.3 Oracle AS 绑定数据时的语法规则

在我们的测试中用到的规则有以下 4 条：

- 全串匹配

如：“submit”翻译成中文就是“提交”

- 嵌套串

如“Edit Policy Result: UniqueCertificateCo”就是由“Edit Policy Result: {0}”嵌套“UniqueCertificateCo”组成的，也要分别绑定后再组合在一起。

- 多串连接

如”OracleAS Certificate Authority-Advanced Search”就是由”OracleAS Certificate Authority”和”Advanced Search”用”-”连接组成的，分别绑定后用”-”连接在一起。

注意：OracleAS 中只存在连接两个串的情况，也就是说即使

prevalue 中有多个”-”，也只有一个是其连接作用的，其他的都是子串内部的。

3.2.3.3 翻译规则的优先级

根据上面的语法规则，我们制定了以下的三种翻译规则类别，优先级由高到低：

1、准备类别

准备类别的优先级最高，是在翻译前必须要判断的，它包括以下两条：

- winrunner 的省略符规则

如果在 prevalue 的首尾发现有“!”和“.*”，则要先去掉“!”和“.*”作为新的 prevalue;然后如果对象的绝对逻辑名包含 prevalue，则用对象的绝对逻辑名作为新的 prevalue

- Escape 掉 prevalue 中所有的单引号

因为最终要用 prevalue 在表的 value 字段上“=”或 like 的 select 查询，而 select 语句中是用单引号作为字符串的边界符的，所以要 escape 掉 prevalue 内部中的单引号，也就是说在 prevalue 内部的每个单引号前再加一个单引号。

注意：这两条规则没有优先级，哪条先执行都可以。

2、精确翻译类别

准备类别后，就是精确翻译类别。包括以下 4 条，优先级从高到低：

- 常量规则

指 prevalue 在常量列表中。

- 精确匹配规则

指用 prevalue 做 select 精确查询成功。

- 嵌套匹配规则

把 GLOBALRES 表中所有满足条件的 value 字段包含有“{0}”

的所有值取出来，把每个值中的“{i}”用“(.*)”代替后做成 pattern，用 prevalue 与每个 pattern 匹配，分别解析出被嵌套的串 outer 和嵌套的串 inner，这样可能有多个解析的结果。要求在至少一种解析结果中，outer 满足精确匹配规则表示成功，而对于每个 inner 串，要先进行常量规则判断和精确匹配规则判断，在不满足这两个规则的情况下，取 inner 串本身。

● 连接串全精确匹配规则

如果 prevalue 中包含“-”，则用不同的”-”解析出所有不同的前后段 tmpfir 和 tmplas。要求在至少一种解析下，满足如下条件时表示成功：

☆ tmpfir 满足常量规则或精确匹配规则或嵌套匹配规则

☆ tmplas 满足常量规则或精确匹配规则或嵌套匹配规则

注意：优先级高的规则先执行，如果成功的话就不再执行优先级低的规则，否则执行优先级低的规则。

3、模糊翻译类别

精确翻译规则失败后，就到了最后的模糊翻译规则。包括以下条规则，没有优先级，都要执行：

● 模糊匹配规则

指 prevalue 做 select 模糊查询成功

● 连接串单精确匹配规则

在多种解析中，至少有一种满足如下条件时表示成功：tmpfir 或 tmplas 中有一个满足精确翻译类别中的任一规则，另一个满足模糊匹配规则或取原值。

● 连接串全模糊匹配规则

在多种解析中，至少有一种满足如下条件时表示成功：

☆ tmpfir 满足模糊匹配规则或取 tmpfir 本身

☆ tmplas 满足模糊匹配规则或取 tmplas 本身

☆ 不能同时取 tmpfir 和 tmplas 本身

注意：翻译规则及其优先级是可以补充修改和删除的，您完全可以根据您的实际情况自己来指定。

3.2.4 对象多翻译的处理

在 3.1.1 小节我们指出了“对象多翻译”情况的存在，如果处理呢？翻译时会把这些对象的所有可能的翻译结果都保存下来，等到输出时再决定取哪个翻译。至于取哪个翻译我们提供了以下两种模式：

● 自动模式

会自动取第一个获得的翻译输出

● 手动模式

☆ GUI 文件

将多翻译的对象单独输出到一个文件中，让用户自己选择。一个典型的输出如下：

"OracleAS Certificate Authority-Certificate Management".Go:

{

class: object,

MSW_class: html_rect,

html_name: 前进

html_name: 下一步

}

☆ Script 文件

不论是单一翻译还是多翻译都输出到一个文件中，对于多翻译的情况，把除第一个翻译外的其他翻译行都用“#”注释掉，让用户觉得以后打开哪个翻译。一个典型的输出如下：

global_image_click("{ class: object, MSW_class: html_rect, html_name: 结束, location:0} ", 22, 10);

#global_image_click("{ class: object, MSW_class: html_rect, html_name: 完成, location:0} ", 22, 10);

#global_image_click("{ class: object, MSW_class: html_rect, html_name: 提交, location:0} ", 22, 10);

3.2.5 多种语言同时翻译的处理

有时我们想从一种语言同时翻译到多种语言，最简单的办法是一种一种的翻译，这样做虽然可行，但是却浪费了时间和资源。事实上，在每种语言下，从 prevalue 到 key 的过程是一样的，结果也是一样的，所以只需要在翻译第一种语言时执行从 prevalue 到 key 的过程就可以了，然后把 key 保存在内存中，在翻译其他语言时直接使用就可以了。这样可以节省很多时间。

3.3 COFAL 的版本

到目前为止，COFAL 有两个版本。

3.3.1 COFAL1.0

COFAL1.0 具有以下特点：

- Client-Server 两层结构的应用程序，使用者要在本地运行；
- 把所有的 ResourceBundle 打包到 jar 文件中
- 用 jar 包直接翻译

3.3.2 COFAL2.0

COFAL2.0 具有以下特点：

- 是 J2EE 架构的 web 应用程序，使用者通过浏览器即可访问；
- 把所有的 ResourceBundle 都导入到数据库中；
- 通过 JDBC 用数据库进行翻译；

与 1.0 相比 2.0 有以下优点：

- 通过浏览器即可翻译，不用自己在本地运行；
- 增加了翻译策略，提高了翻译的成功率；
- 增加了翻译单个 value 的功能，能给出 value 对应的 key

和该 key 在其他语言下的 value，以及这个“key-value”对的出处；

- 增加了翻译 script 脚本的功能

与 1.0 相比 2.0 有以下缺点

- 翻译速度上有一定损失

3.4 COFAL2.0 的使用简介

我们把 COFAL2.0 打成一个 ear 包，部署到一个 OC4J 服务器上，就可以通过浏览器访问 COFAL2.0 了。

3.4.1 翻译 script 或 gui 文件

- 第一步

访问下面这个页面可以开始翻译 script 或 gui 文件：

http://:8990/tina/oracle/cdc/sgt/coral/Trans-step1.uix

在 File Location 中输入要翻译的 script 或 gui 文件，file description 是可选项，点击下一步会把这个文件上载到服务器上。

- 第二步

必选项：

☆ 文件的类型（script 或 gui）

☆ 文件当前的语言和将要翻译成的语言（可多选）

可选项：

☆ 文件涉及的组件（可多选）

☆ 组件的版本（单选）

建议指定组件和版本，这样查询速度会快一些，结果会更准确一些。

- 第三步

翻译前有一个 confirmation 页面，如果选择“yes”则马上开始

翻译，否则回到上一页

开始翻译后，会出现一个等待页面，里面有一个小时钟在不停的转动，真正的翻译通过多线程在后台进行。

翻译结束后，会有一张下载文件的 table 显示出来，选中一个文件，点击“save”按钮就可以把文件保存在本地。

- 第四步

下载文件后，可以选择是否删除原始的和翻译的文件，如果不选择删除的话，可以在当前 session 结束前，翻译其他语言或下载。

- 返回

在这个页面中列出了当前 session 中所有已经上载的原始文件，可以选择上载新的原始文件，也可以选择已有的原始文件进行新的翻译，还可以选择下载已有的翻译文件。

3.4.2 翻译单独的串

访问下面这个 URL 翻译串：

http://:8990/tina/oracle/cdc/sgt/coral/StringSearch.uix

- 在“search for”中输入要翻译的串，可以为任何 COFAL 支持的语言，对于每种语言我们在后台会给它一个缺省的 encoding。
- 在“from lang”中指定串当前的语言
- 在“to lang”中指定要翻译成的语言，缺省为翻译到所有 COFAL 支持的语言。
- 选择在查询范围内的 component 和 release 等，这些都不是必选项

● 如果想把翻译的结果保存在文件中，支持 UTF8 和 native 两种编码方式。

查询结果如下图所示：

4. 利用 WinRunner 做日期格式等本地化测试

日期等格式的本地化测试是全球化测试的一个重要部分，界面上的日期等格式由客户端或客户端浏览器的当前 locale 决定，也就是说在不同的 locale 下日期等都应该有正确的格式。现在的问题是如何在 WinRunner 中实现日期等格式的本地化测试。答案是正则表达式。

下面我们先从定义正则表达式说起。

4.1 定义正则表达式

Oracle 提供的 locale 有 142 个，通常用在以 Oracle 数据库为中心的应用程序中，如 Oracle Locale Builder 和 SQLPLUS 等。

Java 提供的 locale 有 134 个，通常使用 JSP/UIX 页面技术的 Oracle web 应用程序会用到，如 Oracle AS。

下面就给出 java 提供的简体中文 locale 下的一些常用的格式和部分正则表达式。

Date Format	[Date Format] Short 05-6-1 Medium 2005-6-1 Long 2005 年 6 月 1 日 Full 2005 年 6 月 1 日星期三 统一的正则表达式： (\d{2}|\d{4})-\d{1,2}-\d{1,2}|\d{4} 年 \d{1,2}月\d{1,2}日\s*(星期[一二三四五六七])?

	注意：也可以给每种格式分别定义一个正则表达式
	[Date/Time Format] Short 05-6-1 上午 2:03 Medium 2005-6-1 2:03:04 Long 2005 年 6 月 1 日上午 02 时 03 分 04 秒 Full 2005 年 6 月 1 日星期三上午 02 时 03 分 04 秒 IST 统一的正则表达式： (\d{2}|\d{4})-\d{1,2}-\d{1,2} [上下]午\d{1,2}:\d{1,2}(\d{1,2})|\d{4}年\d{1,2}月\d{1,2}日\s*(星期[一二三四五六七]) [上下]午\d{1,2}时\d{1,2}分\d{1,2}妙(IST)?
	[Time Format] Short 上午 2:03 Medium 2:03:04 Long 上午 02 时 03 分 04 秒 Full 上午 02 时 03 分 04 秒 IST
	[Date Symbols] Months(From Jan.) 一月，二月，三月，四月，五月，六月，七月，八月，九月，十月，十一月，十二月， Short Months(From Jan.) 一月，二月，三月，四月，五月，六月，七月，八月，九月，十月，十一月，十二月，

	Weekdays(From Sun.)，星期日，星期一，星期二，星期三，星期四，星期五，星期六 Short Weekdays(From Sun.)，星期日，星期一，星期二，星期三，星期四，星期五，星期六 AM/PM(From AM) 上午，下午 Eras(BC, AD) 公元前，公元
Currency	Currency Symbol:CNY Currency Name(ISO4217):CNY Currency Code:CNY Currency Example ￥123,456,789.00 -￥123,456,789.00 -￥123,456,789.55 正则表达式如下： -?￥[1-9]{1,3}(,[0-9]{3})*(.[0-9]+)?
Number format	Number Example 123,456,789 123,456,789.123 -123,456,789 正则表达式如下： -?[1-9]{1,3}(,[0-9]{3})*(.[0-9]+)? Integer Example 123,456,789 -123,456,789 Percentage Example "0.4" = 40% "0.9999" = 100% "-0.5" = -50% Decimal Separator 。

	Grouping Separator ， Zero Digit 0 Percent ％ Pattern Separator ；
Timezone	[Asia/Shanghai] Display Name: (UTC+08:00) 北京，上海 Short Name CST Long Name(English) China Standard Time Long Name(Native) 中国标准时间 [Asia/Hong_Kong] Display Name: (UTC+08:00) 香港 Short Name HKT Long Name(English) Hong Kong Time Long Name(Native) 香港时间

有一点需要特别说明，我们下面给出的这些格式只涵盖了简体中文 locale 下的部分情况，具体要用到哪些格式需要定义哪些正则表达式，要由您要测的应用程序来定。

4.2 自定义 re_match 函数

WinRunner 自带了一个叫做 match 的函数，用来判断一个串是否符合正则表达式。但是，WinRunner7.5 中的 match 函数对过于复杂的正则表达式的支持不是很好，鉴于日期等的正则表达式的复杂度比较高，我们使用了一个由 EMOS Framework 提供的已编译过的函数 re_match()。

这个新的函数实现了一个类似于 perl 语言中正则表达式查询和匹配功能，弥补了原来 match 函数的不足。

```
public extern._int re_match(in._string str,
                            in._string re,
                            out._int m_pos,
```

out._int m_len,

inout._string detail)

说明：

将串与正则表达式做匹配。输出 detail 中的子匹配结果可以通过另外两个函数 re_get_details()和 re_get_match()获得。

参数：

str－ 要匹配的串

re－ 正则表达式

m_pos－ 匹配发生的开始位置

m_len－ 匹配的长度

detail－ 匹配的细节

返回值：

0 = 不匹配, 1 =匹配， m_pos 和 m_len 分别表示匹配的开始位置和匹配的长度

下面是一段使用该函数的代码：

input_string="2005 年 6 月 1 日";

pattern = "(\\d{2}|\\d{4})-\\d{1,2}-\\d{1,2}|\\d{4}年\\d{1,2}月\\d{1,2}日\\s*(星期[一二三四五六七])?";

if(re_match(input_string, pattern, m_pos, m_len, detail)){

= tl_step("Locale sensitive date format check", 0 , "Date format is OK");

treturn("PASSED");

}else

{

tl_step("Locale sensitive date format check", 1 , "Date format is incorrect");

treturn("FAILED");

}

注意：这只是一个简单的例子。实际应用中，input_string 和 pattern 都可以参数化到文件，在不同的 locale 下读取不同的文件动态获得，以保证脚本的通用性。

关于 EMOS Framework，它以区别与其他 Framework 的独特方式实现了一个 WinRunner 自动化脚本开发的简单但却强大的框架，EMOS 是专门为 WinRunner 设计的，整个 EMOS 几乎都是用 WinRunner 自己的 TSL 语言实现的。关于 EMOS Framework 的详细介绍请参考 http://emos-framework.sourceforge.net/。

小节

至此，您应该对用 WinRunner 实现软件的全球化测试有了一定的了解。我想强调的是：本文的重点不是想象您推荐我们开发的 COFAL，而是提供一种思路。这种思路的意义不仅在于它部分地实现了自动化全球化测试的目标，更在于它为测试人员打开了一扇门，工具是死的，关键在于使用工具的人，如果工具做不到的，我们可以想办法让它做到。也许在寻找过程中您会收获到对软件更全局化的理解，收获开发人员对我们的尊敬和咨询，最重要的是收获您自己对这份工作的肯定和信心。

参考文献

- Sun 关于 Internationlization 在 JDK 里面的说明
  http://java.sun.com/j2se/1.3/docs/guide/intl/intlTOC.doc.html
- SUN 的国际化教程
  http://java.sun.com/docs/books/tutorial/i18n/index.html
- WinRunner 的用户手册

终端的性能测试分析

作者：黄向东

摘　要：本文对终端产品的性能测试提供一些技术介绍。结合本人工作实践，对终端性能测试需要涵盖的范围进行了分析，然后对终端性能测试的自动化测试方法进行了探讨。

关键词：终端性能测试，自动化测试

进入2005年，国产手机厂商利空消息频频传来，这和市场格局变化导致国产手机厂商在原先的一些渠道、价格、品种等方面的优势逐渐丧失有关，但是还有一个不容忽视的原因是国产手机的质量问题过多，产品稳定性差，突出的问题有：自动关机、死机、掉线、通话质量差、杂音大等。这些问题中的大部分都是可以通过充分的性能测试来避免或减少的。

手机作为专用的消费类电子产品需要进行以下测试：可靠性测试（对于硬件则是RQT；对于软件则是field trial）；标准符合性测试（FTA）；互操作性测试（IOT）；安全性测试（安规测试）；强度测试等。

其中，有些种类的测试，例如FTA，有严格的标准（GSM、3GPP等）来明确被测的功能点，测试人员所要做的是在测试用例的编写中体现出这些功能点，并且尽量营造这些测试用例所需的运行环境来完成测试，并反馈测试结果。但是对于性能测试，就没有这样的规范供测试人员来参考，因此性能测试需要进行哪些用例以及用例通过的指标的高低都有很大弹性，在很大程度上受限于测试人员的经验和项目的资源和进度压力。如何在资源、进度和质量之间找到平衡点是产品负责人需要考虑的问题，测试人员可以左右的是划定性能测试的范围、明确与性能测试相关的设计需求（提高产品的可测试性）以及通过自动化测试工具等手段来进行更加有效的性能测

试，提高产品的质量。

一、手机性能测试的范围

性能测试强调长时间、重复或者高强度的进行某些操作，来验证产品在各种极限条件下的表现。性能测试隶属于软件测试中的系统测试，它对软件在集成系统中运行的性能行为进行测试，旨在及早确定和消除软件中与构架有关的性能瓶颈。通过对测试数据和 log 的分析，还可能找出被测系统隐藏的缺陷。终端作为移动通讯类电子产品，其性能测试又主要和其实现的功能相关，大致可分为以下几类：

i. 时间相关。

时间相关的性能测试可分为长时间保持测试和限定时间反应测试。

长时间保持测试主要是测试终端长时间稳定进行某项功能的能力。主要包括长时间待机能力、长时间 CS 域业务保持能力、长时间 PS 域业务保持能力、长时间组合业务保持能力等。长时间待机测试，就是根据手机电池的能力连续不间断待机一定时间（例如 4 天），之后验证手机是否还能够发起主叫和被叫业务，能够发起主叫，表示终端在长时间待机后自身还处于正常状态，能够发起被叫，说明终端在睡眠模式下可以正常接收寻呼。长时间 CS 域业务保持测试，就是根据手机电池的能力连续不间断进行语音通话或者视频通话一定时间（例如 2 小时），测试通话期间图象声音是否连续、清晰，是否有单通现象出现，是否会有手机板子过热现象。长时间 PS 域业务保持测试，主要是通过持续进行 WWW 业务、ftp 业务或者流媒体业务一定时间（例如 2 小时），测试进行数据业务期间上下行数据传输率是否稳定，网页显示是否流畅，流媒体播放是否连续等。长时间组合业务保持测试，就是同时保持 CS 和 PS 域业务一段时间，以验证终端长时间进行组合业务的能力。

限定时间反应测试主要是测试终端在规定时间内对用户的操作

作出反应，给出操作结果的能力。主要包括开机驻留时延、关机时延、CS 域业务接入时延、PS 域业务接入时延、本地应用的操作时延等。开机驻留时延，是指从用户按下开机键（终端上电、系统引导、启动任务、搜索网络、完成位置更新）到终端进入待机界面，提示用户可以进行正常服务的总时间。关机时延，是指从用户按下关机键（终端完成网络 detach、将 RAM 中修改过的数据写回 flash）到终端完全下电所需的总时间。CS 域业务接入时延，是指在进行语音或视频电话时从按下拨号键到听到对方回铃声所需总时间，由于该过程需要在网络侧分配资源，所以测试结果可能会受到当前网络资源可用程度的影响，例如在网络负荷高的时候申请 CS 64k 业务时，网络侧需要重新组织或合并无线资源来满足业务要求，所需时间相对会长一些。PS 域业务接入时延，是指在进行数据业务时从开始连接到能正常进行数据业务所需总时间。本地应用的操作时延，是指完成某些本地操作维护功能所需的时间，例如打开电话薄，在电话薄里查找联系人，存储新建的联系人，存储短信，存储多媒体文件，打开浏览器，播放多媒体文件等所需时延，这些时延如果过长，也会极大地降低用户体验的满意度。

ii. 次数相关。

次数相关的性能测试是测试终端重复稳定地进行某项功能的能力。包括开关机成功率、小区初搜成功率、小区重选成功率、CS 域业务成功率、PS 域业务成功率、组合业务成功率、切换成功率、本地应用的成功率等。这种重复操作包括很多对象被多次创建和释放，因此可能会发现潜在的内存泄漏等问题。开关机成功率测试，主要是检验多次开机是否会有物理层不能正确收到初搜命令的情况，关机不完全也可能会导致下一次开机失败，以及在某些情况下系统死机后只能通过插拔电池板来重新开机。CS 域业务成功率的测试，是指通过进行一定次数的主叫或者被叫，统计失败的次数，对失败原因进行归类，分析是否能够找到和终端相关的失败原因。PS 域业务成功率、组合业务成功率、切换成功率的测试方法也类似。本地应

用的成功率包括多次存储再删除文件、联系人、短信等操作，以及多次打开某个应用或执行某类操作来对该应用的稳定性进行测试，找出瓶颈。

iii. 并发业务。

并发测试主要是测试终端同时进行多项业务时表现出的处理能力。例如同时进行 CS 域语音业务和 PS 域下载业务，或者在 MP3 播放的同时进行 WWW 上网业务，以测试协议栈、操作系统和处理器对并发业务的支持能力。

iv. 负载测试。

负载测试主要是验证系统的负载工作能力。系统配置不变的条件下，在一定时间内，终端在高负载情况下的性能行为表现。例如同时进行多个 ftp 下载，使下行传输率接近极限值，观察终端是否可以正常工作。

二、手机性能测试的方法

手机性能测试的方法按照自动化程度不同可分为手工测试和自动测试。

手工测试主要是通过测试人员手动操作，并借助某些监测仪器和工具，来验证手机性能。但由于手机功能众多，并且性能测试工作量大，如果单个测试工程师靠手动按键来执行所有测试用例，花费的时间少则几小时，多则需要几天的时间，这样耗费大量测试时间的同时也容易让测试工程师产生疲倦甚至是厌倦心理，很容易造成测试的遗漏。手机测试中常碰到很多重复性高的工作，如发送数条 SMS 或者 MMS 以验证其收发成功率以及稳定性、连续进行多次呼叫、多次对文件系统进行添加删除操作、多任务多进程情况下的冲突测试以及极限测试等等，都是重复性高的工作，手动执行的话费时费力，如果能有一套自动执行的机制，将能大大提高测试的效率。

由此产生了对手机自动化测试工具的需求。手机这种板机的

MMI 功能测试不同于基于 PC 上的 MMI 测试，后者借助 PC 平台，目前市场上已有非常多功能强大且通用的自动测试工具支持其测试，如比较典型的有 Winrunner， Robot， Loadrunner 等等，但这些工具通常不能兼容到象手机这种嵌入式系统中来。这就要求测试人员能够基于当前平台进行二次开发，来满足自动化测试的需求。

手机的自动化性能测试一般分为以下几个步骤进行：

1. 系统分析

将系统的性能指标转化为性能测试的具体目标。通常在这一步骤里，要分析被测系统结构，结合性能指标，制定具体的性能测试实施方案。这要求测试人员对被测系统结构和实施业务的全面掌握。

2. 建立虚拟用户脚本

将业务流程转化为测试脚本，通常指的是虚拟用户脚本或虚拟用户。虚拟用户通过驱动一个真正的客户程序来模拟真实用户。在这一步骤里，要将各类被测业务流程从头至尾进行确认和记录，弄清这些过程可以帮助分析到每步操作的细节和时间，并能精确地转化为脚本。此过程类似制造一个能够模仿人的行为和动作的机器人过程。这个步骤非常重要，在这里将现实世界中的单个用户行为比较精确地转化为计算机程序语言。如果对现实世界的行为模仿失真，不能反映真实世界，性能测试的有效性和必要性也就失去了意义。

3. 根据用户性能指标创建测试场景

根据真实业务场景，对生成的测试脚本进行复制和控制，转化为满足性能测试指标的测试用例集。在这个步骤里，对脚本的执行制定规则和约束关系。具体涉及到对业务类型，并发时序等参数的设置。这好比是指挥脚本运行的司令部。这个步骤十分关键，往往需要结合用户性能指标进行细致地分析。

4. 运行测试场景，同步监测应用性能

在性能测试运行中，实时监测能让测试人员在测试过程中的任何时刻都可以了解应用程序的性能优劣。系统的每一部件都需要监测：协议栈，MMI 应用程序，内存占用情况，驱动程序运行状态等。

实时监测可以在测试执行中及早发现性能瓶颈。

5. 性能测试的结果分析和性能评价

结合测试结果数据，分析出系统性能行为表现的规律，并准确定位系统的性能瓶颈所在。在这个步骤里，可以利用数学手段对大批量数据进行计算和统计，使结果更加具有客观性。在性能测试中，需要注意的是，能够执行的性能测试方案并不一定是成功的，成败的关键在于其是否精确地对真实世界进行了模拟。

在整个性能测试过程中，自动化测试工具的选择只能影响性能测试执行的复杂程度，简便一些或繁杂一些；但人的分析和思考却会直接导致性能测试的成败。所以这里着重于对性能测试思路的整理。测试工具的介绍可以参看有关自动化测试工具的资料。

我们应该向谁学习

作者: Tender

在论坛上晃了很久了，每一天都会看到各式各样的贴子，其中绝大多数的贴子都是一些新手发出的。既然论坛作为一个交流的平台，每个人都可以发表不同的意见和感慨。当然，也少不了争论和谩骂。IT 人总体上说的确比较浮躁，其实说得更准确一点，我们这一代人都比较浮躁，容易冲动，容易上火。

论坛上几乎每天都有朋友在问该怎么学习，时间一长，有些人就不耐烦了；也有表达自己观点的人，结果持反对意见的网友们开始与其辩论了。说辩论是好听的，经常在论坛上晃得朋友们应该都知道，这种贴子过不了几天，就变成骂街的贴子了。

有时候我也一直在想，怎样帮助那些求助的朋友们？怎样才能真正让他们理解学习的涵义？除了书本和实际工作以外，还有哪些值得我们去学习？怎样学习？向谁学习？

华为现象——向企业学习

我在论坛上用“华为”作为关键字搜索了一下，找到相关的帖子有三页之多。可见，华为已经成为我们关注的一个话题。对于华为的评论是褒贬不一，有的是听别人说华为怎么样怎么样；有的是亲身走出华为的人说华为怎么样怎么样；有的人说好，有的人说差。那么，华为到底怎么样？它的利弊在哪里？现在的年青人是否应该

去华为？华为到底告诉了我们一些什么呢？

先来看看华为的成绩：1988 年才起家的华为，在任总的带领下，经过短短 18 年的发展，已经成为国内外数一数二的无线通信网络设备供应商，目前正专注于 3G（WCDMA/CDMA2000/TD-SCDMA），NGN，光网络，xDSL，数据通信等几个领域，全球员工 3 万多人。2003 年，华为销售额 317 亿元人民币，2004 年实现销售额 462 亿元人民币，其中海外销售额 22.8 亿美元，2005 年技术销售额收入 453 亿元人民币。当前，世界电信运营商前 50 强中，华为已经进入 28 家，除中国运营商外，还进入了英国电信（BT），沃达丰（Vodafone），西班牙电信（Telefonica），荷兰 KPN，新加坡电信，泰国 AIS，南部非洲 MTN，巴西 TELEMAR 等著名运营商。据 Dittberner 统计，华为 NGN 系统全球市场占有率 18%，全球排名第一；交换接入设备全球出货量连续三年位居第一；据 Gartner 统计，华为 DSL 出货量全球排名第二；据 RHK 统计，华为光网络市场份额全球排名第二；华为是全球少数实现了 3G-WCDMA 商用的厂商，已全面掌握了 WCDMA 核心技术，并率先在阿联酋，中国香港，毛里求斯等地区获得成功商用，跻身 WCDMA 第一阵营，成为全球少数提供全套商用系统的厂商之一。2005 年 11 月，华为与全球最大的移动通信运营商英国沃达丰正式签署了全球采购框架协议，华为将参与沃达丰的移动网络建设。目前，华为的 WCDMA 正在印度尼西亚的 NTS 开局，而且华为正在为进入全球另一个最大的移动通信运营商 T-Mobile 做出不懈的努力。

华为的成绩还有很多，有朋友会问：没有缺点吗？我说有，没有一件事情不存在两面性的。

最敏感的问题就是薪水了。不错，华为全球已经有了 3 万多员工，这还不包括其子公司的员工。华为已经不可能像以前那样用高

薪去吸引人才了。我记得5，6年前，应届毕业生进入华为都能拿到5K+的薪水，奋斗2，3年后就可以买车买房了。而现在的毕业生只能拿到3K+，而且只能进入慧通。说到慧通，也是很多人不愿去华为的原因，感觉在慧通，低人一等，干一样的活，薪水却很少，福利也不多。于是，一个薪水，一个慧通，把华为的威望降下了许多。其实，这里面存在着很多恶性循环，我举一个例子：华为现在招聘员工，薪水压低，新员工先进入慧通，而这些条件已经不那么吸引人了，特别是吸引不到真正的牛人。于是，现在进入慧通的员工整体水平下降，很多技术性很强的工作无法胜任，很多老员工非常不满，于是又去招人，条件又很低，又是招了很平常的员工，老员工又不满，完成不了工作接着再去招人。于是，人是越招越多，质量却越来越低，以前一个人干两个人活，现在两个人干不了一个人的活，还抱怨薪水少，辞职的辞职，调换的调换，弄得新人的整体水平不断地下降。

慧通的成立的确是为了降低成本，储备人才，这也是企业的生存法则。可是，昔日的华为给人们留下了太深太深的印象，以至于一点点的变动，就会引起很多人的不满。坦率地讲，公司大了以后，都会变得官僚，节奏会慢下来，会多元化，丧失了创业公司的活力，这是必然的，就像小孩子长大之后就不会活蹦乱跳了。

利与弊都说了，我们到底要向华为学些什么？要向企业学些什么？

无论是多有名的企业，都会存在不足之处。我想，华为之所以成功，更关键的是它有很多做得好的地方。我认为，如果你是一个刚毕业的学生，如果你对自己的前景还比较迷惘，如果你能去华为，那么就去华为！

去华为学习它的企业文化，学习它的精神。让我最难忘的是它的群狼精神。华为是一头狼，是一头地地道道中国式的土狼，可就是这头土狼，一跃冲出了国门，冲向了世界。无论你在哪个部门，哪个产品，你都能切身感受到狼的野性。所有员工配合工作，团队之间的合作以及部门之间的合作，到处都是那种群起而攻之的景象。2006 年 1 月 8 日至 13 日，华为的 HAJJ 项目组在沙特麦加朝圣期间取得了圆满成功。这一套服务于沙特电信的 GSM 移动软交换设备经受住了考验，全球各地的 300 多万移动用户，集中在 5 平方公里的范围内，同一时间使用手机进行呼叫，话务量之高达到前所未有。难怪无线产品线总裁张顺茂先生说：“成功经受麦加朝圣的话务高峰再次证明了华为移动软交换技术的先进性和稳定性。” HAJJ 项目成功了，可背后的付出正是大家不懈努力换来的。一月底的时候，我有幸参加了华为无线产品线的迎春晚会。在会上，华为执行副总裁徐直军和无线产品线总裁张顺茂都对 HAJJ 项目给予了高度的评价，无论是否在一个部门，是否在一个产品，全场员工都在为他们呐喊，这种凝聚力，是无法用言语能够描述的，若不是亲临现场，是无法能够感受到的。有谁能够忘记创业时的艰辛，有谁能够忘记开拓市场时的忙碌，有谁能够忘记电信行业的那个冬天。可是，华为就这样一步一步挺过来了。这是一种精神，是一种企业文化在激励着每一位华为人！

去华为学习它的技术，学习它的流程。华为研发普遍实施 CMM 管理，印度，南京，上海研究所和中央软件部都通过了 CMM5 级国际认证，北京研究所通过了 CMM4 级国际认证。从 97 年起，华为开始系统地引入世界级管理咨询公司，建立与国际接轨地基于 IT 的管理体系，在集成产品研发（IPD），集成供应链（ISC），人力资源管理，财务管理，质量控制等诸多方面，华为与 Hay Group，PWC，FhG 等公司展开了深入合作。无论你从事哪个方面，在华为都能学习到很好的流程和管理理念。

我在这里不是夸华为，我只是想告诉朋友们，我们应该理性地评论一家企业，学习它做得好的方面，同时也要看到它的不足之处。关键问题是，如果我们有幸能进入类似华为，中兴，联想，甚至是微软，IBM，Intel，普华永道，毕马威这样的大企业，我们先要问一问自己，去这些好的企业干什么？是去享福还是去学习？是去每月无所事事等着拿不错的薪水还是踏踏实实地去干一番事业？如果你想明白了，就不会因为暂时的低薪而埋怨；如果你想明白了，就不会因为环境的恶劣而丧气；如果你想明白了，就不会因为一时的困难而沮丧。

向企业学习，学习它的成功之道。

偏激现象——向中庸学习

曾经在论坛上看到这样一个贴子，一位网友问是学日语好还是学韩语好？结果有些急性的朋友马上回答：学汉语好！于是又是一大片争论，最后变成谩骂。

类似这样的贴子还有很多，这其实就是偏激现象。我们先不说学哪个语种好，其实这就像一个辩题，学哪个语种都有其自身的道理，就算争个鱼死网破，最后也不会下个定论说到底学哪个好。大家都看过辩论赛，正反双方就自己的观点能说上一大堆的道理，可最后结果出来后，我们能说今天正方赢了，所以正方的观点是正确的，明天反方赢了，所以反方的观点是正确的，能这样说吗？正反双方的论点本身就是辨正地存在于社会中，就像汉语和外语一样。不会因为今天你有理，所以大家都学习汉语，把外语弃之国门之外，那不是又变成闭关锁国，夜郎自大了吗？也不会明天你有理，所以

大家都学习外语，把汉语丢弃了，那不是又变成崇洋媚外了吗？其实，我心里也明白，要求先学好汉语的朋友们心里多少带着对日本的仇恨。

民族感情是要的，但不要什么事情都举着民族感情的旗帜。如果我们一看到日本就讨厌，那么岂不是又陷入了另一种偏激之中——民族歧视。60 多年前，日尔曼民族歧视犹太民族，几乎赶尽杀绝；大和民族歧视中华民族，屠杀了多少中华儿女。这是他们犯下的罪行，我们应该引以为鉴，不能重蹈覆辙。如果哪一天，我们也有了民族歧视，那后果会是什么样子？从历史上来说，大元朝统治中国的时候，蒙古人到处杀戳汉人，结果汉人反了，推翻元朝，建立了大明。后来呢，汉人又到处去残杀蒙古人，弄得汉蒙两族人民活在仇恨之中。大清的时候，满人当政，汉人辅政，满汉共同治国，和平友好，才出现了康乾盛世。现在国家始终实行民族区域自治政策，要是有民族歧视，国泰民安也只是妄想徒劳！

孔孟学说中有“中庸”之道，意思是看任何一个事务，干任何一件事情，不要极端。就像上面提到的学习哪个语种一样，如果我们都理性地分析一下，就不会说出那么偏激地言词来了。中庸不代表懦弱，而是一种为人处事地方法。偏激容易使对方跳，而中庸才会让对方笑。

中庸之道是一种哲学，把握它需要长期地磨练才能达到一定的境界，不要异想天开地希望在短时间内就能成功。

向中庸学习，学习它的哲学之道。

职业规划现象——向行业学习

虽然论坛是一个测试领域的交流平台，但我相信一定有很多研发的朋友在论坛上和我们一起交流，也有以前做过研发现在转行做测试的朋友。

关于做研发好还是做测试好的话题，论坛上也有过争论，这倒也没什么。让我心里很不爽的是，刚发了一个贴子说测试很有前途之类的话，紧接着楼下的贴子就说，测试有什么了不起，做不了研发才做测试，你们做测试的当然说做测试好。老王卖瓜，自卖自夸！

我只想说，又是一个偏激主义，又是一个行业歧视！研发做不了才做测试，典型的鄙视测试行业，这就好像在说做不了医生才做护士，当不了飞机长才做了技术支持。

我们先来看看测试是不是“鸡肋”。既然测试不如研发，那也就可以说护士不如医生，技术支持不如飞机长。有人说，对啊，是不如啊！那我要问，哪里不如？医生能够动手术，那少了护士行不行？医生能够诊断，那少了护士的日常护理数据行不行？机长能开飞机，那少了技术支持行不行？我么可以假设一下，在动手术的时候不要护士，反正医生也知道下一步是拿剪子，还是镊子，还是纱布，自己心知肚明，何必要护士再当个“二传手”，还要隔着厚厚的口罩说活，万一听不清楚拿错了怎么办？驾驶飞机的时候不要技术支持，机长一个人开呗，出现故障了就自己捧着两，三千页的全英语技术文档看呗，查完了排除故障了再接着开飞机，有什么难的？朋友，这现实吗？如果可以执行的话，我想有关部门早就实行了吧，还可以节约人力成本，何乐而不为呢？

无论是医生和护士，机长和技术支持，还是研发和测试，都没有主次之分，也没有孰优孰劣之分。大家站在不同的岗位上，履行

着不同的职责，相互配合，相互合作，才能把一项工作做好，过分强调其中一个而贬低另一个，都是不合理的。就像我们身上的五官，少了哪一个都不行。所以，测试绝对不是食之无味，弃之可惜的“鸡肋”。

我们再来看看研发。研发人员为什么一直都瞧不起测试？因为他们一直以来都认为研发在整个软件生命周期中是最重要的。可惜，这个观点又是错误的。研发是重要的，但不是最重要的。

我姑且把软件生命周期分为六个阶段：需求分析，概要设计，详细设计，编码，测试和维护。一直以来，许多人都认为编码在其中占有很大的比重，而且事实也如此。后来，出现了软件危机，这才提出了科学的软件工程概念。于是，大部分的软件产品把重点放到了需求和设计阶段上，但此时编码还是占据了比较大的比重。再往后，随着各式产品的不断出现，质量问题逐渐浮出水面，于是测试又开始受到了重视。纵观一下可以发现，无论哪个阶段开始被重视，编码一直都是占据很大席位的一个阶段。

可真正成功的产品是怎样一个划分比例呢？50%以上的时间都应花在需求和设计上，编码阶段只有 10%－15%，测试阶段 20%－30%，余下的是维护阶段。有的人马上会反驳，研发人员每天没日没夜的写程序，难道只占 10%－15%？不要惊讶，一个成功的项目就是如此。试问，研发人员整日整夜写代码的项目成功了吗？产品交付后研发人员轻松了吗？为什么现在很多研发人员不愿去只会写代码而不要流程的公司呢？不要认为测试占 20%－30%是瞎扯，好的测试流程从产品立项到项目关闭，一直处于工作状态，每个阶段都要做好测试工作，才能保证整个产品的质量。所以我说，这个比例还是少的。现在研发平台已经越做越好，为研发人员节省了很多时间，甚至好的工具在画出 UML 图后就可以根据 UML 自动将其转

换成代码，这样更是节省了研发人员再去敲键盘的时间，所以编码在整个软件生命周期中所占的比例是越来越少，不是不重要了，而是工作效率提高了，工具先进了，有更多的空余时间了。于是又有研发人员说，我们又有更多时间去歧视测试了。但我要说，原先编码在整个生命周期内占的比例高，所以你们看不起这个看不起那个。现在编码的整体比例下降了，你们还有什么资格去看不起别人！？

最后看看行业的发展。研发人员一直说，研发好，有前途，月薪多少多少，升职了还能有多少多少。测试嘛，冷门行业，没人干的，没什么前途的。试问，什么叫冷门行业？没人做的行业叫冷门，没人要的行业叫冷门。那么什么时候有人要，什么时候没人要？5，6 年前，机械专业没有人学，整个机械行业停滞不前；两年前，机械自动化，机械一体化突然兴起，刚毕业的学生供不应求。上世纪九十年代中期，国际贸易火得不得了，可现在呢？还有会计行业，企业管理行业，现在都已经趋于平淡。谁又能预言研发会一直火下去而测试会一直冷下去？毫无疑问，测试行业刚刚起步，国内的人才还比较少，而研发已经兴旺了很多年。大家不妨去了解一下，现在招聘研发人员是什么条件？招收人数是多少？我想比几年前差远了吧，要求高了，人数也少了。2005 年，CSDN 做了一项调查，研发人员的从业经验。一年以内的有 1511 人，1－3 年的有 2736 人，4－5 年的有 1133 人，6－10 年的有 537 人，11－20 年的有 54 人，20 年以上的有 3 人。可见，目前研发人员的经验主要集中在了 0－3 年之间，数量之多也是显而易见的。经过这几年的发展，底层的研发人员已经趋向饱和，而真正缺乏的是那些架构师和系统分析师。而这些职位的要求又是非常之高，以至于一般的研发人员都是望尘莫及。反观测试行业，近几年刚刚起步，供不应求，发展空间非常大，要求也不是很高，所以入门比较简单，整个行业处于一个上升阶段。

其实，我只想反驳那些歧视测试行业的人，并不是看不起研发。

虽然编码阶段的整体比重在下降，但研发人员也一直在维护阶段为修改问题，实现新需求而忙碌着。无论做哪个行业，都有其自身的优势和劣势。研发行业要求高，学习范围比较广，但是人数太多趋向饱和，怎样才能使自己从底层的程序员走向架构师和系统分析师是研发人员要考虑的问题。同样也应该看到，测试行业虽然刚起步，门槛低，但缺乏有经验的测试人才，国内培训没有及时跟上，企业内测试流程的不规范，这些也都是成为优秀测试人才的瓶颈。

行业之间也是应该互相学习的，没有一个行业能说自己永远是最重要的，最优秀的。只有不断的学习，相互的帮助，才可以共同进步，共同繁荣。

向行业学习，学习它的生存之道。

最后，我还想告诉各位网友，理性地去看待每一件事情，每一个人。抓住每一次学习的机会，无论学习的对象是一个人，一家企业，一种品质，还是一种道德，一种精神。

真诚地祝福所有 IT 人，事业蒸蒸日上！

二零零六年元月初六

一个测试新手与 mercury 认证的不解情结

作者：黄碧澜

2005 年 9 月 26 日，不知天高地厚的我，参加了上海 51testing 组织的 Mercury 认证考试，因为感觉性能测试是软件测试行业中待遇比较高的职位，为此本人也在这种拜金主义心理的作祟下，报考了 loadrunner 这款工具。10 月 19 日，51testing 的朴老师电话通知我考试成绩——74 分，哈哈，顺利通过！谢天谢地，俺这样的新手第一次参加这样的考试，竟然通过了！感谢 51testing 老师们的辛勤培训，应该说，他们提供的考前辅导还是相当有效的；当然也要感谢自己，三个星期来的日夜兼程，把 loadrunner 这款工具的使用搞的死去活来，挥洒自如（有点自夸哦）。

据 51testing 的老师讲，本季度考试通过率不错，在 50%左右，尤其上个季度没有通过的考生，这次通过了很多。为此，本人笔兴大发，决定总结一下自己的考试经历，哈哈，大家给点面子，别乱拍砖，照顾下俺这个测试新手吧！

从哪说起呢？既然想总结下自己这 1 年多来的工作生涯，干脆从毕业说起吧！我是个北方人，毕业于西安某三流的大学；北方人一直对北京都有个莫名的情结，所以 2004 年毕业后只身闯京城，拿着计算机本科毕业证，在举目无亲的都市里漫无目的的穿梭于各大招聘会。一个月下来，没有任何单位有要我的意思；唉，都要工作经验，而且我又不是名校毕业，眼瞅着和几个清华、北工大的家伙一起去投简历，都不好意思和面试官打招呼，唉，可怜的我！没办法，只好想办法钻点空子！经历了十来场招聘会，总算摸清这些单

位的用人需求，不就是 java 么？不就是数据库么？不就是 unix 么？好，咱学！于是在北京 7 平米的小茅屋里寒窗苦读，把这些上学时候学的稀里糊涂的东东搞定，于是迎来了某软件公司的初试——做题！（如果大家有过类似经历，我会和你报头痛哭；如果没有，ggjj 们，你们是不知道现在的用人单位招人多严苛，上来啥也不说，先做 2 小时的卷子再谈！）还好，这次准备充分，而且几个月来做的卷子不少（当然都被“莫需有”的理由拒了），别的能耐咱没有，笔头子功夫练出来啦！结果笔势完毕，在门口外和一个北邮的家伙一起抽烟等待通知。这哥们和我有着相同的被拒经历和被拒次数，因此马上情趣相投起来，我手上的烟也是他派的，一看，是我讨厌的“中南海”！没办法，不抽白不抽！这哥们和我一样来应聘程序员，看着他饱含沧桑的眼角，我深深体会到：1）这年头，和我一样的毕业生，找工作难哪！2）这小子抽烟手势不端正，因此经常呛到眼睛，哈哈！

半小时后，一个打扮比较风骚但是一点都不漂亮的中年女人（之所以把她先拉开帏幕，因为她就是我日后的上司），和另外一个大汉（他是日后我们项目的开发经理）出来，叫我俩进去一下，对其他来笔试的人又说了点漂亮的拒绝言辞；这时候，我又深深意识到两点：1）我和北邮兄有戏 2）那些被拒的弟兄将会继续走完我走过的路…

第二轮面试时，我和北邮兄只是在一个小屋里被不漂亮女人和大汉简单的蹂躏下；最后的结果是：北邮兄由于 java 基础好，去做了程序员；我呢，去做测试！在离开小屋的那一刻，我下意识的悄声问了下北邮兄：啥叫测试啊？北邮兄：好像是测试程序吧！前边的大汉似乎听到了我们的对白，回头用极其鄙视的眼神看了我一眼，那眼神锐利得直刺我的胸口，好像对我说：小样，——新来的吧！

日后的岁月里，我在这个小公司极其混乱的测试流程里煎熬着。

虽然本人不聪明，但是咱笨鸟先飞，在 7 平米的小屋里，无数个黑夜里，俺懂得了软件工程，俺了解了 RUP，俺知道了测试应该按阶段来实施，不是想测什么就测什么，俺还知道测试用例的非常重要，但是不知道为什么公司里做测试不按照测试来执行呢？于是俺上网找资料，希望把自己的想法说给女人听，俺想让她知道：俺不想一直拿 1800 的试用期工资，俺想改进公司的测试流程，俺想离开 7 平米的没有暖气的茅屋，因为冬天来了，俺不想被冻死…3 个月后的年底转正，我写了一份“热气腾腾”的工作总结递交给女人，想好好表现一把。但是俺失败了，在女人威严的慑服力下，俺屈服了！因为后来我和北邮兄讲，他告诉我：如果要采纳你的意见，这个公司的老板该回家当奶爸了！

此后是我 3 个月的人生低潮期，因为俺的转正工资只有 2300，扣了社保不到 2000；因为俺依然要在茅屋里蹲过这个冬天，因为除了填饱肚子，所有的积蓄都要还给北邮兄，买个破手机要我还 3 个月，唉！黑夜里，仰望夜空，很美；唱着杜甫的《茅屋为秋风所破歌》，很凄凉！我不知道这样的日子要持续多久，不知道是公司不容我还是我容不下公司，不知道自己该何去何从！

这次应该感谢 51testing 的 sincky！为什么呢？因为他的一篇文章把我从这段低谷里挽救了出来。年后的某日，在网上偶然看见一篇文章《借刘德华演唱会小喻测试人生》，开始看着感觉写的挺诙谐的，后来深思了一下，感觉说的很有道理：原来没有人可以一步登天，原来要成就大事就要从小事做起，原来这么浅显的道理却被很多人忽视，竟然不能化为实践！在他的 blog 里，我学习到很多，也因此在这个时候和他在 msn 里结识，原来每个成功的背后都隐藏了几多心酸和挣扎，因为在很多经历里，我们竟然有几分的相似。于是，我意识到自己是时候该奋起直追了，蹉跎了岁月，伤透了年华，既然国内的软件测试行业如此的不成熟，是不是说明我有很多成功

的机会呢？在网上结识了几个测试领域的高手以后，我又一次认识到自己的不足，因为：从前我懂的测试理论、测试流程是一种标准化的东西，对于国内很多软件公司来说，还仅仅是一个方向而已！这么说来，我没有能力改进公司的测试流程，但是我有能力改进自己的测试技能！相信很多同行朋友从前或目前一定有和类似的心境，那么在你读到这里的时候，你一定该立刻振作起来，从自身出发！换句话说，如果每个测试从业者个人的能力都提高了，整个软件测试行业就自然的进步了。

这时候，就不得不提自动化测试了。应该说，刚工作的时候，我就搜索各大软件测试的网站、论坛，发觉自动化测试是每个论坛里讨论最多的话题。开始由于自己没有软件，而且感觉学自动化测试的朋友们总有那么多问题，所以感觉这东西可望而不可及。但是有几次和北京业内的测试朋友见面聊天的时候，他们都一致认为自动化测试是测试行业发展的方向，而且这东西可以自己掌握，不需要改变公司的任何规章，哈哈！于是，从朋友那里拷来一份 mercury 的 winrunner、loadrunner，还有个 rational robot（好像一提到自动化测试工具，想到的基本就是这几个，看来我知道的都晚了!），趁下班的时候，在自己的机器上玩命的摆弄。开始也没想太多，反正别人用了，自己也不甘落后，不然总觉得别人一谈自动化测试，自己没的说，感觉不爽！起初，学习的时候，有问题到论坛查资料，后来发现 51testing 的论坛资料还是非常全的，于是统统下载来，成天研究手册，然后在机器上练习，把自己的机器搞的乌烟瘴气，不过从来没有让女人发现过，强吧！

开始，有问题我会到各大论坛去提问，但是后来发觉 51testing 的提问响应时间是最快的，这是让我第一次佩服 51testing 的做事效率，后来干脆就不去别的论坛了。由于自己有点 java 编程基础，加上 java 和 c 语言差不多，我就从 winrunner 开始学起；于是花了 2

个来月，我把 winrunner 学完了，怎么叫学完了呢？因为 2 个月后，我已经从原来的提问问题请求别人回答，过渡到后来的看别人问题给别人回答了！其实大家不要拍砖，因为此时虽然我能给别人解决一些论坛里的问题，那是因为大家学习的时候，遇到的问题和我遇到的都差不多，所以自己多思考一下就成了；其实那时候我的工具使用水平还很差，直到后来我参加 51testing 的远程培训才真正意识到这种差距还不是一点点。

那个时候，我和一些论坛的朋友一起学习使用 winrunner，那种攻破一个个技术难关后的成就感，是我们很多搞技术的朋友们经常会有的，很满足，很兴奋！winrunner 学罢，自然而然的就想学 loadrunner；为什么呢？因为他们是一个公司的东东，而且脚本方式基本类似，另外，我也渐渐知道 loadrunner 作为全球市场占有率最大的性能测试工具的强大优势，加上经常浏览 51job 上的测试工程师招聘信息，让我更加坚信这一点：如果我能把 winrunner、loadrunner 都学好，那么离开眼下的这个公司是迟早的事。私下问了一些朋友，在北京做自动化测试的职位月薪有多少，统计了一下，在 3500 到 6000 左右，如果是外企或者用友、神码这样的公司，做好了会更多；另外一个值得庆幸的是，性能测试工程师有的企业标价在一万以上，呵呵，这不，有奋斗目标了不是！但是自己清楚目前资力还很低，很多东西还是学习阶段，没有真正在企业里实践过，所以还是放住脚跟，把现有的知识学扎实了。

3 个月后，loadrunner 基本使用搞定了；下班的时候，试着给公司的服务器加压，哈哈，有一次真的把数据库搞 down 机了，第二天公司的系统工程师查到是我的 ip 搞的，于是来质问我究竟想干什么；我说做性能测试啊！它听说是用 loadrunner，结果反倒跟我请教问题，哈哈！这小子总认为自己做网管不爽，反倒羡慕我们做测试的，说这是一个长期的饭碗；网管么，青春饭（这是他的原话，诸位网管

朋友别拍我)!

这个时候发生在今年夏天，由于自己能力的飞跃增涨，这种压抑的物质欲望也愈加膨胀；加上在北京认识了很多业内的高手，他们开始建议我跳槽了…怎么办呢？说到要离开，还有点舍不得这里的弟兄们，北邮兄仍然风雨兼程的闷头苦编程序，偶尔一起抽烟吃饭；但是他是北京人，工资问题不是他的最大问题（顺便提下，他的最大心愿是想拥有一份超过 3 个月的完整恋情，哈哈；因为每个女朋友都是莫名其妙的一个电话、一个短信就提出分手了!)。但是网管哥们劝我跳槽要谨慎，要跳就跳个像样的公司，如果只为了几百块钱，去熟悉一个全新的公司，有点划不来。他说的不无道理，但是想让自己的待遇翻倍谈何容易？再征求多方朋友的意见下，最后决心努力做好学好 loadrunner 使用，在性能测试领域寻找突破口。要做性能测试，光有工具的使用远远不够，要把这款工具真正的用到实际软件的性能测试中，要对软件架构、数据库、操作系统、网络都要了解；好，说干咱就干！孤家寡人一个，没有别的夜生活，没有别的娱乐活动，除了学习就是拿公司的软件练习（感谢我原来的公司，真的，虽然女人、大汉也许都不知道此事，这里还是要感谢他们的默许和睁只眼闭只眼)。

7 月份，我偷偷的面试了一个北京某外包公司，在问到性能计数器的添加方面，我哑口无言；8 月份，被问到模拟一个真实网站性能测试的场景设计有什么注意事项时候，我又无言以对…唉！怎么办？我所学习的仍然很基础，实践性的东西我还很欠缺！要找一份性能测试工程师的理想职位，还真是难啊！

这个时候，不得不重提 51testing；因为此时我得知 51testing 推出了 mercury 工具的国际认证，而且提供工具使用培训和考前培训。开始我对认证并无兴趣，只是想参加他们所说的 CPE 工具培训，来

多学点东西，从而弥补自身摸索阶段的不足。于是我报名参加了51testing的loadrunner CPE培训，这时，是我第二次佩服51testing老师的能力过人之处；原来在CPE培训里，我曾经遇到过的一些困惑和疑难，经过老师的讲解，真有豁然开朗之感，而且从他们那里获取的一些资料，让我对性能计数器的使用有了更加清晰的认识。

之前因为sincky的那篇《漫谈测试工程师与mercury认证》的文章，和他交流过；但是当时觉得国内的认证做的太烂了，而且自己一向认为能力不是靠证书来证明的，是良马终究会遇到伯乐的！但是在经过CPE培训后，自己由衷感觉到他们的培训实力还是相当强的，毕竟是给企业做培训的老师，讲出来的东西确有高人之处；可能是自己被他们的人格魅力折服了吧，呵呵！于是心里暗暗打算，将来有钱了，是不是也去考一个SP或者CPC啊，哈哈！

一个偶然的机会，在51job上发现一条某公司的招聘信息，性能测试工程师，月薪12000，再看招聘条件的最后一项写着“mercury认证者优先”。我赶忙瞪大眼睛，再看是一个外企，不错、不错！看来我这条路算是选对了，自动化测试——有搞头！于是重新温习了sincky的那篇《漫谈测试工程师与mercury认证》，起初认为文章说的比较有道理，但是依然认为有一些商业宣传的味道，但是现在想来，文章说的还是相当有道理的：

1）“觉得国内的认证做的太烂了”：和51testing的人接触过后，特别是在后来我去上海参加SP的认证考试，和51testing的老师们面谈后，发觉他们是真真正正的在做软件测试的培训，真真实实的想把MECRURY认证做好；有这样精干、高效、强劲的团队，从我个人角度讲，这个认证的价值我相信！

2）“认为能力不是靠证书来证明的”：这话到现在我也不否认，

个人能力不是一张纸能体现出来的，即便是纯金的，即便是比尔.盖茨亲笔签名的；但是证书却是获得高薪工作的敲门砖，尤其在这个证书还没有普及到人手一份的时候，它更能体现你在众多竞争对手时候的优势！为什么呢？因为我有证书，所以我有自信！因为我有证书，在心理暗示上，我认为我对工具的使用熟练度上要比别人强！从另一个角度来说，国内学习自动化工具的人成千上万，但是真正能熟练使用的人又有多少呢？也许这话不该出自我这个新手之口，但是在我后来到上海面试某大型软件公司的性能测试工程师时，深刻体会到我身边的那边朋友与我的差距（如果那位朋友看到此文，在此非常抱歉，不过我没有任何贬低你的意思，呵呵），所以我觉得如果能通过认证考试的人，起码对工具的使用是非常熟练了，从mercury 认证的考试范围你会发觉它考的方面相当广泛，不是轻易就能通过的。

3）“是良马终究会遇到伯乐的”：这话也不假，不过从现在严峻的就业行情来看，“酒香不怕巷子深”的岁月已经不复存在了，想找一份理想的高薪的工作，不是自己努力争取，没有人会送上门来找你；即便是猎头公司上门来拉你，那也是因为个人能力到达一定境界。但是想要达到这个境界，突破个人职业生涯的瓶颈，就要付出很多的，当然包括这种类似的智力投资啦！

哈哈，不分析这个啦，拙见、拙见！于是我联系了 51testing，准备报名所谓的 loadrunner 的 SP 认证考试，当然在价格上，他们也给我打了折扣；虽然花去了 5 个月的积蓄，但是我知道迟早有一天，我会翻倍的赚回来。9 月底在 51testing 的远程培训站点上，俺认真研究了每个模拟题，了解了考试的难度和题型，之后在 9 月 24 日抵达上海，参加 26 日的考试。其实这时候，我已经电话联系好一个上海的那个大型软件公司，27 号参加面试。该公司购买了 loadrunner 工具，并招聘性能测试工程师，准备组建自动化测试团队对公司开

发的软件产品进行性能测试。面试期间没有什么特别剧情要说给大家，只是他们听说我参加了 mercury 认证考试，加上我回答 loadrunner 使用问题上的自信表情，所以同意录用了我；非常欣慰，因为我即将告别北京的那所 7 平米茅屋，因为我的薪水翻了 2 倍多，哈哈！

十.一七天就这样结束了！明天我就要背上行囊，踏上上海的征程了。独自走在灯火璀璨的京城长街上，回想起 1 年来的生活片断，那种百感交集的感觉是无法用文字来形容的；就要离开了，虽然那里是另一个物欲横流的社会、另一个繁华喧闹的城市，但是我仍然怀念你：北京——这个城市，还有更重要的是这个城市里的好朋友们，包括女人和大汉；女人，其实你的妆不化那么浓，还是比较漂亮地；大汉哥，我的项目经理，其实你该经常刮下胡子；还有北邮兄，感谢你经常派给我不爱抽的“中南海”，不如尝尝“白沙”；网管兄，希望早日在测试行业里见到你，不过怕是见不到你了，因为你都 36 啦！…

我的文笔并不好。本来由于刚刚获悉认证考试的结果，兴奋过了头，所以想写点东西总结下自己；因为对我而言，这次认证考试的经历过后，让我进入了一个全新的职业生涯里程，所以想和大家分享一下；结果话说多了，把这 1 年多来的经历都流水帐的记录出来了，没办法，本人爱罗嗦是出了名的！既然本文标题是关于 mercury 认证的，那么最后请允许我再给将要报考 mercury 认证的朋友一点经验分享：1）SP 等级的考试一点都不难，但是考的内容很细，它是真正的考核我们的工具使用熟练度；所以一定要在考前多多的实践，练习它的每个功能点的操作 2）考试题目虽然是英文，但是都是很简单的英文，连我这样的新手都能一次通过，大家该更有信心才对 3）外国人出题很单一，绝对没有怪僻的题目，甚至它在题设里就告诉你，应该有几个答案，所以没有什么迷惑性 4）题目难度不大，但是题量比较大，而且是总分的 70%算及格，所以答题时候

要谨慎哦，使遍浑身解数也要争取把题目做对哦 5）51testing.com" target="_blank">51testing 组织的 mercury 认证考试，每个考生都可以有 2 次考试的机会，所以我觉得即便第一次失误没有通过，有了一次经验，在第二次肯定可以通过的，只要你用心去学习它 6）也是最重要的，就是要珍惜 51testing 提供的考前培训，虽然培训里没有原题，但是这是初次应考者唯一的考前辅导，价值不可忽视

自动化测试的职位，在网上火的大红大紫，希望这个势头多持续几年，哈哈，来日等我考取 CPC 认证，就可以去应聘那些高级的性能测试工程职位喽，起码让自己的投资有所回报，也尝尝月薪过万的感觉！朋友们，让我们互相祝福吧！

就这样吧，做为一个新手，很多话讲出来怕同行笑话，不过俺也不怕啦，自己的道路自己选择，同意我的说法的、不同意我的说法的，都可以抽空来和俺畅谈；初到上海，正缺朋友聊天呢，呵呵！联系 qq 411469763，不过刚刚来到新公司，不能经常在线，见谅啦！