第5章　高级进程、触发器和跨平台设计

第4章讲述了用Oracle Developer做的许多工作，本章研究这个产品的内部工作情况，展示一些特殊的进程，以及它们是如何工作而产生所想要的效果的。有了进程的知识，在某些细节上，就可以了解如何使用 Oracle Developer的事件处理特点，为以后的几章中的高级教程作准备。

注意　本章的辅导和例子给你说明如何以及在哪里的触发器上使用PL/SQL编程，将你自己的逻辑加到Oracle Developer的标准进程中。通常，这些标准进程就足够了。第4章讲述了如何完全不用编码建立一个工作应用程序。第11章和第12章详细地讲述PL/SQL编程。所以本章只是简单地提供示例的代码，而不去进一步充分讲述编程的细节。这些示例说明如何完成触发器的基本任务，你可以用在这里和在第 11章第12章学到的东西为基础，去建立非常复杂的应用程序。

在Unified Modeling Language(UML)中，活动的框图表示进程。活动框图通过进程表示工作的流动，很像旧的流程图。但是活动框图直接涉及对象的状态和它们之间的控制流。它们适合于说明Oracle Developer运行时的表单、报表和图形的进程工作流。在本章的第一张图中给出了一些例子，图中的文本足够评细地对活动图进行解释，使你能了解它。本章讲述在Oracle Developer中跨平台开发应用程序的一些有关问题。展示了要得到最大的可移植性如何设计表单、报表和图形。

5.1　Oracle Developer进程和触发器：它们是如何工作的

这一部分综述Oracle Developer是如何施用它的魔法的。前面几章介绍了 Oracle Developer数据库应用程序和对象的基本概念，本章讲述这些对象如何相互作用，象一个系统那样工作。

在设计Oracle Developer应用程序中，最实际的低级设计是决定将代码放在哪里。因为你不是根据Scratch来开发应用程序逻辑，所以必须使增加的东西适合于 Oracle Developer的逻辑。为了做到这一点，需要知道 Developer在哪运行你的代码，并在执行代码之前和之后发生什么。这一部分通过说明这些进程是什么，在这些进程内的什么地方激活触发器，清楚地告诉你在Oracle Developer进程的什么地方可以进行干预。

Oracle Developer应用程序是事件驱动的。当程序运行时，你和运行系统相互作用引发事件，事件又启动各种表单的进程。这些表单进程包含有预先编程的表单、报表和图形的缺省特性。在各种内部事件发生的过程中，运行时的系统通过运行你用 PL/SQL写的代码来处理它们：激活触发器。

你可以在高层次上继承设计师在这个产品中建立的一组假设和特性，而不必从头开始分析和设计。用一组预先编程的特性开始，这是一个很高的开发起点。本章的进程描述了 Oracle Developer应用程序的缺省运行时的特性。了解这些进程帮助你像 Oracle Developer那样思考，而不是根据Scratch来开发整个系统特性。如果你利用 Oracle Developer工作，而不是与它对着干，那么，大概会成功。

5.1.1 表单进程和触发器

在Oracle Developer的三个部分中，Forms有最成熟的逻辑。这部分概述Forms中不同的进程和触发器。

首先，必须了解一个基本的概念：导航和范围。

1. 导航

用户界面允许系统的用户工作。虽然这似乎是明显的，但其重要性在于了解表单开发结构的基本原理。Oracle Developer的表单是事件驱动的，这就意味着用户通过用户界面来控制进程，而不是应用程序来控制它。

在任何给定时间，Oracle Developer Forms的用户界面允许用户和单个对象相互作用。比如按钮、下拉列表(菜单)或文本域。单个对象是用户界面的一部分，是用户的焦点 (focus)。引用焦点的另外一种方法是引用游标的位置 —— 一种虚拟的指针，它指到有这个焦点的对象。光标是虚构的野兽，有时允许人看见它。例如，当游标出现在文本域时，它通常作为对平台输入文本的指示器，比如Windows中的细直条或Macintosh中的一束射线。其他项，比如图片或单选按钮并不明显地显示光标，所以使用“焦点”这个术语可以避免混淆。特别是术语“游标”至少指三种不同的东西，它可以指鼠标，SQL的游标(cursor)，或独立于表单的键盘游标。

注意 “单个对象”指的是单个应用程序对象。从表单进程的角度看，用户界面，比如窗口和菜单是不相干的，只有画布上的项目可以放置焦点。然而从用户界面的角度看，“焦点”像对对象一样，也适用于包含应用程序焦点的窗口。所以，如果焦点在文本域，那么，它也在包含文本域的画布的窗口上出现，而且也在应用程序的 MDI窗口出现。在这本书中，术语“焦点”指的是应用程序对象焦点，不是窗口的焦点。你可以从McGraw-Hill Web站点，www.Osborne . com和这本书的例子一起下载这些进程的综合参考材料。

用户通过从对象到对象移动焦点与应用程序交互，这个移动就是 navigation(导航)。通过从项到项、从记录到记录、从块到块，或从表单到表单导航，用户控制应用程序和任务。导航事件(进入到不同的项，进入到不同的记录，进入到不同的块，进入到不同的表单)构成了表单应用程序的基本事件。后面部分的许多进程或者包含这些事件或者是这些事件的结果。

在Oracle Developer Forms中导航的对象按导航单元的层次排列。表单包括一系列的块，每个块包含一组记录和一系列项。Oracle Developer Forms的导航活动是由于用户的活动而自动发生的导航。如果焦点在一个块的一个项上，而在这个块的另一个项上单击，那么则从项导航到项。如果项不再同一个块上，那么必须从数据块导航到数据块。如果每个数据块有不同的记录，那么你还必须从记录导航到记录。每一个导航事件都有不同的结果。比如某些触发器启动(在这种情况下，分别是When-Validate-Item、When-Validate-Block和When-Validate-Record(当确认项时，当确认块时，当确认记录时)。

对象的层次 —— 项、记录、块、表单 —— 导致一组进程，关系到导航的层次，比如说Navigate to Form Level (导航到表单层)。在这些表单进程的名字中的“层”(Level)指的是在对象层次结构中的导航单元：

- Navigate to the Form Level(导航到表单层)
- Navigate to the Block Level (导航到块层)

■ Navigate to the Record Level (导航到记录层)

■ Navigate to the Item Level (导航到项层)

每一个表单进程都把焦点从当前单元移到指定的目标。这个进程识别包含目标和“ enters ”的导航单元的下一个比较小的子单元，把焦点设置到那个对象上。

导航进程自动地确认当前的导航单元。例如从某一项导航到表单层，那么根据导致这个导航的特定事件来确认这个项。某些事件导航不确认，大多数导航有确认。大多数取决于单元 —— 层，在这层上通过表单的 Validation Unit(确认单元)属性设置需要确认的表单。本章后面的“ 确认 ”部分，讲述基本的确认进程。

虽然大多数表单导航是用户行为的结果，但进程也可以引起导航发生。例如在许多 Oracle Developer Forms触发器中，可以调用内部的子程序，比如 Go_Item、Go_Block.或Go_Record。在本章的后面，“ 导航的处理 ”比较详细地讲述导航。

注意 导航是Oracle Developer的关键概念。实际上，所作的每一件事情都和导航有关。虽然这个概念和进程其本身是相当简单的，但它们普遍存在，与许多有意思的而有时是不可预测的方法相结合，超过任何别的东西。在 Developer中导航是必须在字面上和图形表现上均要“ 照此办理 ”的，试图“ 对付 ”Developer的导航结构，总将发生问题。

2. 触发器作用域

可以按照Object Navigator的分层结构在项目层、块层或表单层定义 Oracle Developer Forms的触发器。触发器作用域是一组激活触发器的对象，它由拥有触发器的对象和属于这个对象的对象组成。例如，如果把一个触发器连接到一个块，那么这个块的所有项的事件都启动这个触发器。

注意 留心的读者可能发现，最后一个段落并没有提到记录层作用域。虽然有记录导航触发器，但并不是在记录上定义这些触发器，记录只在运行时存在，在Form Builder中它不能作为对象得到。因此，不能把触发器连接到“ 记录 ”上。因为数据块可以看作是记录的聚集，所以经常将面向记录的触发器连接到数据块上。这些块触发器由当前记录 —— 有焦点的记录 —— 启动，因此记录触发器没有实际的作用域。

如果在一个特定的作用域内有多个触发器有相同的名字， Oracle Developer按缺省值启动连接到层次结构中最低的那个对象的触发器。例如，如果把一个 When-New-Item触发器放在一个项和这个项的块上，那么 Oracle Developer启动连接在项上的触发器，而不理会连接到块上的那个触发器。可以通过改变触发器属性表中的 Execution Style(执行方式)属性改变特定触发器的属性。缺省值为 Override。还可以选择Before或After，Before规定触发器在任何较高级的触发器之前启动，而 After则是在较高层的触发器之后启动。次高层的触发器可以选择和当前的触发器一起启动。可以在较高层的作用域层上设置这些属性，使得在这些层上的所有的触发器可以在项目触发器之前或之后启动。

提示 利用Execution Style(执行方式)设置为Before或After使多个触发器启动，显著地增加了模块的复杂性。除非有十分充足的理由要这样做，否则应避免它，以使模块比较容易理解、调试和测试。

某些触发器只有在特定的层上定义它们才有意义。例如， When-Validate-Record并不适用于单独的项，只可以在块和表单层上定义它。 SmartTriggers特性识别那些很可能为特殊对象

定义的触发器。在Object Navigator或Layout Editor上右击想定义触发器的对象，引出弹出菜单包含一个SmartTriggers项，在这个项上单击，显示一个子菜单，列出对于这个对象最有意义的触发器。选择适当的项，显示这个触发器的 PL/SQL编辑器。子菜单还包含一个 Other项，显示标准的触发器选择对话框，可以列出所有的 Oracle Developer触发器(见图5-1)。

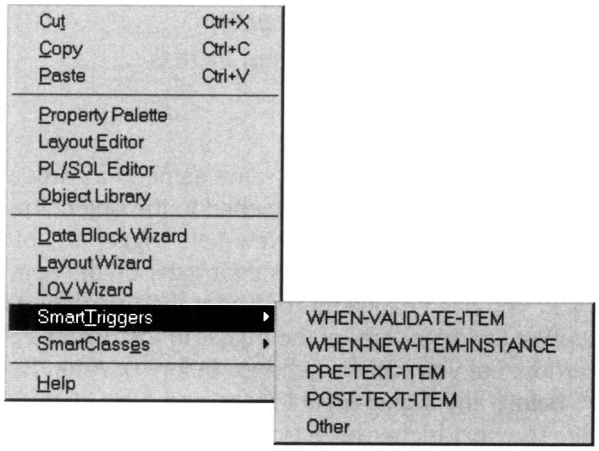

图5-1 Oracle触发器

也可以通过属性类继承触发器，而且可以引用已经定义过的触发器。这些触发器有一个相同的规则：仅有一个触发器执行。如果用项上的触发器取代 Property-Class触发器，则项触发器执行。对象组仅能够定义表单层的触发器，块和项的触发器和它们的块和项一起产生。

注意 为了简单起见，本章的图表不展示触发器和它们的判定。但是每一个触发器都会成功或失败，如果触发器成功，进程如流程图所示的那样继续，如果触发器失败，进程停止，带着错误返回。

现在已经对导航建立了一个概念，而且基本了解了触发器是如何工作的，如何利用SmartTriggers，以及如何修改触发器的缺省属性。下面该了解进程和触发器在实际的项目中都做了些什么。

3. 事务处理

表单事务是一连串的事件、进程和触发器，最终导致将数据提交给数据库或回滚变化。这一部分讲述事务中实际工作的基础进程。下面包括一些真实的进程。

1) 登记数据库　登记数据库的意思是通过一系列的 INSERT、UPDATE和DELECT语句把表单中正在进行的各种变化写进数据库。 Oracle Developer产生这些语句作为它的缺省处理的一部分。登记并不提交这些变化，但是它通常伴随提交或回滚作为缺省性的一部分。登记进程的作用是启动所有有效的触发器，所以每一次对数据库的登记都使存贮的数据生效。不管是否提交。

在缺省情况下，在数据库的一次登记后面是结束事务的提交或回滚。但是，可以利用POST内部程序来登记变化而不用提交。不用提交的登记的主要用法是，当从另外的表单打开一个表单时，可能在第一个表单中有未提交的数据，而第二个表单的存在可能正是为了加更多的数据到一个正在进行的事务中。

例如，可能要在一个会计事务中输入一项，这要求通过一个单独的表单在一个辅助表上

加一些详细的信息。在打开第二个表单时， Oracle Developer要求未完成的数据都要记入数据库以保证数据的完整性。因此，应调用项目的 Post in the when-Validate-Item触发器代码。

注意 第10章详细讨论多表单处理以及管理它的不同技术。

在详细讨论登记过程之前，对活动流程图作一简略的说明。图形中长方形是行为说明，说明内部的行为和至少一个转移。本章的活动图全部用内部的 Oracle Developer Forms进程作为行为说明，比如Savepoint 或Clean up cursors。箭头表示转移 —— 从一个行为说明转移到另一个行为说明或判断上。有两种特殊的转移，起源于黑圆圈的转移是一个空转移，表示图中活动的开始。在外带一个圆圈的黑圆圈处终止的转移是一个退出转移，表示活动结束。退出转移是“ 成功 ”还是“ 失败 ”取决于活动的结果。菱形表示判断，根据条件分支转移 (如果条件为真，流向一个方向，如果条件为假，流向另一个方向)，在判断处的一个转移旁边的文本是隔离(guard)条件。

图5-2是记入进程的简化活动图。

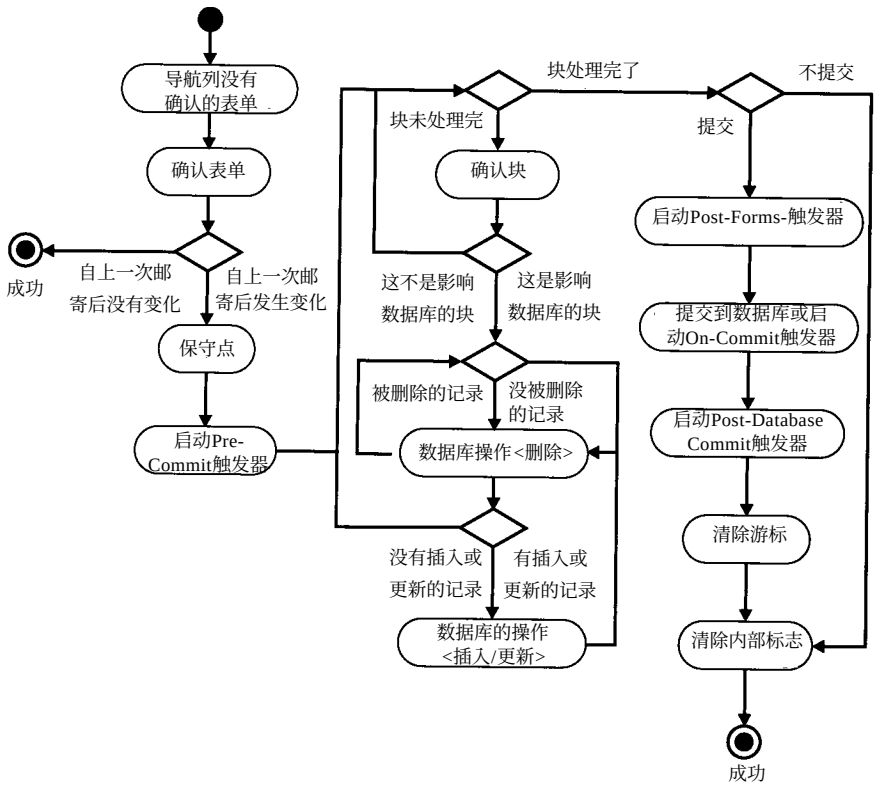

图5-2 记入过程的简化UML活动图

现在，详细描述记入过程。首先， Oracle Developer导航到表单层，然后确认表单，如果没有要求记入的变化，那么过程就停在这里。

然后Oracle Developer在记入之前发出一个保守点，保守点是 Oracle的下一个特性，它标识事务的一个点，如果需要你可以回滚到这个点，而不必回滚整个事务，这就允许 Oracle Developer记入多次而不用回滚所有的记入 —— 只要失败的那个邮件。

注意 除了Oracle，其他的数据库程序没有SAVEPOINT语句。所以如果要求数据库可移置，那么就不应该使用POST程序。而且在利用Open_Form或Call_Form内部函数调用另外一个表单时，Oracle使用SAVEPOINT。所以如果在调用的表单中提交失败，它只回滚到这个保存点。

在发出这个保存点以后， Oracle Developer启动Pre-Commit(预先提交)触发器，然后通过数据块循环。确认每一块，然后确定这个块是否有影响数据库的任何变化。如果这个块是个控制块，就没有这样的变化，如果这个块的基础表是一个不可更新的视图 (带有一个连接或其他不合格条件的视图)，那么只有在它上面定义了事务触发器，Oracle Developer才处理这个块。如果这个块没有变化的记录， Oracle Developer跳过这个块。

块处理将进程分两部分：通过在数据块记录中的循环的方式，处理被删除的记录行和处理记录的插入和更新。图 5-3展示了数据库操作过程的细节。数据库由 DELETE、INSERT、UPDATE语句和Pre-以及Post-operation触发器组成。如果需要控制数据库操作的话 (例如，使用不同于Oracle的、有不同语法的数据库管理程序)，可以用On-<Operation>触发器替代缺省的操作。

处理完所有的块以后，如果进程是提交，而不只是登记，那么 Oracle Developer启动Post-Form-Commit触发器，提交这些变化 (或者启动 On-Commit触发器)并启动 Post-Database-Commit触发器。然后像所要求的那样，清除 SQL游标，然后Oracle Developer清除内部标志，包括把所有插入的记录标记为数据库记录，把任何变化的项标记为 Valid(有效)。

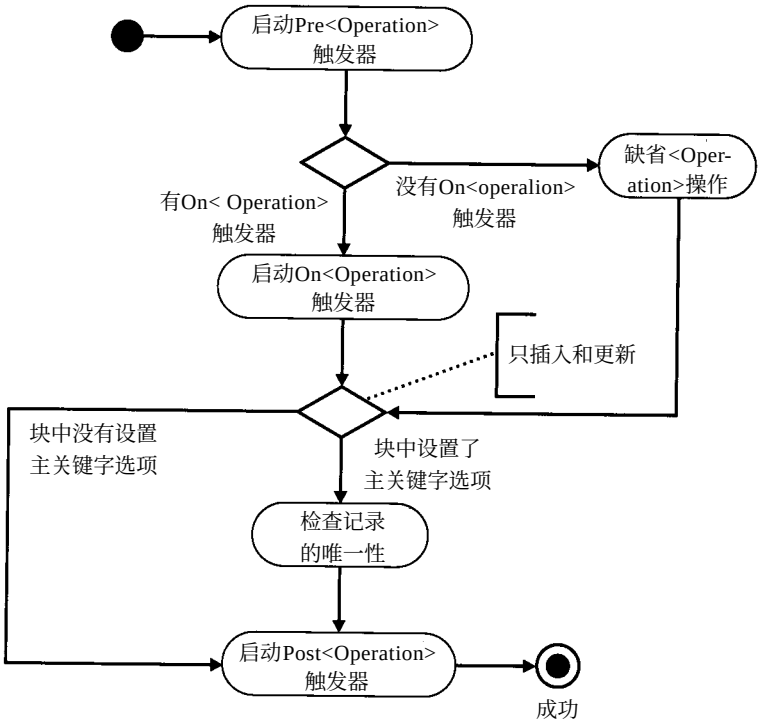

图5-3 数据库操作的UML活动图

注意 如果使用任何一个Pre或Post-Operation触发器来改变表单中与数据库有关的数据，

Oracle Developer可能不确认它们就把这些变化提交给数据库，因为它立即把每一样东西都标记为Valid(有效的)。当触发器启动时，重新确认可能会导致无限的确认循环，造成新变化，并要求更多的确认。

登记中最普通的触发器是Pre-Insert。Pre-Update、Pre-Delete、Post-Insert、Post-Update和Post-Delete(插入之前、更新之前、删除之前、插入之后、更新之后和删除之后)触发器，这些触发器包含有你根据要登记到数据库中的变化而写的代码，这些代码用于检查多表条件或操作其他表的数据，也可能包含有加到应用程序以跟踪特定表活动的检查更新。

注意 Pre-和Post-触发器的许多行为可能更适合于在服务器触发器里编码。就是说，可能想为操作前后的行为在数据库中加触发器。当这些触发器必须引用数据库内不存在的数据时，比如控制块的项值，或还没有移到服务器数据库的其他数据块的值，应该把这些数据移到应用程序的触发器。即如果需要在PL/SQL代码中引用一个块和项，那么可能需要在应用程序中编写触发器代码。使用这些触发器的另外一个理由是把信息处理从数据库服务器移到应用程序，使得到更好更有用的信息成为可能。

例如，Ledger表单让Talbot农场的员工在每天的日记中输入事务，在删除一行之前， Pre-Delete触发器包含有代码查寻用户关于删除事务的安全访问等级。如果用户没有合适的安全等级，则触发器失败，系统显示一个错误信息，告诉用户 (他或她)没有删除事务的特权。

```
DECLARE
  v_Level NUMBER = 0; -- lowest security level as default
BEGIN
  SELECT SecurityLevel
    INTO v_Level
    FROM User_Security_Level
   WHERE User = :Control.Username AND Object = 'Transaction';
  IF v_Privilege < 5 THEN -- delete only for security level 5 and over
    Message('You must have level 5 security to delete transactions.');
    Form_Trigger_Failure;
  END IF;
END;
```

2) 锁定 Oracle数据库管理程序通过锁定资源(比如锁定表和行)，允许同时访问数据而确保其完整性。锁定是Oracle实现事务的方式或工作逻辑单元。某些数据库应用程序锁定整个表，另外一些则锁定数据的物理页。 Oracle Developer使用专门的行锁定，即在特定的行上锁定，这种锁定防止其他事务获得这一行，防止它们更新或删除这一行。在 Oracle 的SQL SELECT语句中加FOR UPDATE OF子句，就需要获得行锁定的信号。

按照缺省设置，在以下任一条件下， Oracle Developer获得行锁定：

- 当操作员改变数据库行中的基础表项的值时，包括从 NULL(空)值变化到non-NULL(非空)值。
- 当操作员通过Record | LOCK (记录|锁定) 菜单项为当前记录要求明确的行锁定时。
- 当触发器通过UPDATE(更新)或DELETE语句对数据库进行修改时。
- 当触发器调用 DELETE_RECORD、ENTER_QUERY(FOR_UPDATE)、EXECUTE_QUERY(FOR_UPDATE)，或LOCK_RECORD内部函数时。
- 当触发器包含显示的 SQLLOCKTABLE语句时。通常，这将严重影响应用程序事务的性能，如果没有绝对充足的理由，没有经过了解该事务的人对设计的专门检查，不应

该使用它。

如果Oracle Developer不能立刻获得锁定，那么它将在固定时间内反复试图获得锁定，然后询问是否前进。如果这种情况经常发生，那么请数据库管理员监控数据库弄清楚为什么得不到这个资源。通常，这是使用了明显的独占表锁定的结果，应当尽可能避免这种情况。

和任何事务一样，当事务提交或回滚时，数据库管理程序释放所有的锁定。因为 Oracle Developer使用保存点，在回滚后可能保留一些锁定，允许在进行必要的变化之后，只重做回滚的部分。

注意 如果在触发器明确地锁定查询行，那么在提交或回滚以后，需要在这些行上重新建立锁定来查询这些行。清除一个块或记录并不释放这些记录行所对应的行锁定。

如果在某一层定义一个On-LOCK触发器，那么每当Oracle Developer试图在这个层上获得锁定时，它就启动这个触发器。可以利用这个触发器来替换缺少的锁定机构 (如果有足够的理由这样做的话)。

警告 在对Oracle数据库运行应用程序时，避免使用On-LOCK触发器，因为可能由于替换了Oracle的锁定策略而降低事务的吞吐量。

通过块上Locking Mode(锁定方式)属性来控制锁定。在创建块时，这个属性的缺省设置是Immediate(直接的)，这意味着如果改变记录的基本表项， Oracle Developer将自动锁定当前的记录。可以把它改变为Delayed，意味着Oracle Developer等待到提交前才获得锁定。在某些情况下，当在查看一个值时，允许其他用户修改它，因为这可以改善大容量应用程序的事务性能。这种行为的风险是，在用户提交之前所做的任何工作被废除，会引起错误，被要求作一些意想不到的附加工作。如果它经常发生的话，可能引起混乱和令人气愤。在因特网上使用应用程序时，最好是利用 Oracle Developer服务器的锁定和事务管理能力。使用这些特性的应用程序开发者不需要担心在他们的应用程序中管理锁定的困难和产生事务管理问题。

另外一种属性是在文本项中锁定记录。通常，对于那些不是基础表项的文本，就是说，那些和基础的数据库表的项不对应的文本项，在使用这个属性时，改变它们并不实行直接锁定。因为并没有真的改变数据库数据行的任何项。如果要改变项的文本，那么将这个文本项的Lock Record属性设置为Yes，即通知Oracle Developer锁定当前的记录。

4. 确认

“确认”是一个过程，是确定一个对象是否满足在定义表单时加在它上面的所有约束。在需要时， Oracle Developer自动确认项目、记录、块和表单。可以在触发器上附加另外的约束来作为确认触发器过程的一部分，或作为事务处理的一部分 (参看前面的部分)。当离开对象或当按下ENTER键或调用内部输入程序时，进行确认。当用户提交时或登记代码改变时登记过程自动确认。

(1) 确认单元和状态

可以确认项目、记录、数据块或表单对象，按照缺省设置，确认项将自动确认，这意味着每当离开项目时，即产生确认。可以设置确认单元为记录、数据块或表单，可以设置在它们的某一层自动确认。如果是这样， Oracle Developer在所规定的层上产生确认动作，而在此之前不做任何确认。例如，如果选择的是 Form，则当运行表单的确认并且登记变化时才确认。通常，为了得到最好的结果，应该让确认单元在项目这一层。可以通过表单对象的 Validation

Unit 属性改变确认单元。

提示 Oracle公司建议，只有当用不同于Oracle的数据库管理程序运行应用程序时才改变Validation Unit属性。

在研究动态特性的细节之前，需要了解在 Oracle Developer Forms应用程序中与触发器有关的项和记录的不同状态。对象状态是在对象生命周期中某个点上的对象属性的设置。大多数Oracle Developer对象都有很多不同的属性，比如屏幕位置、宽度、高度等。与触发器有关的对象状态并不一定指的是这些属性，而只是指和触发器有关的特性。

下面的内容分别从构造单独的项一直到构造表单。每个对象均涉及其他的对象状态，通常按照层次向下构造。

(2) 项目的状态和确认

项目的状态必须由项目的值以及对这个值做了什么来决定。项目可以是一个新的、变化的或有效的。图5-4展示了确认过程的活动图

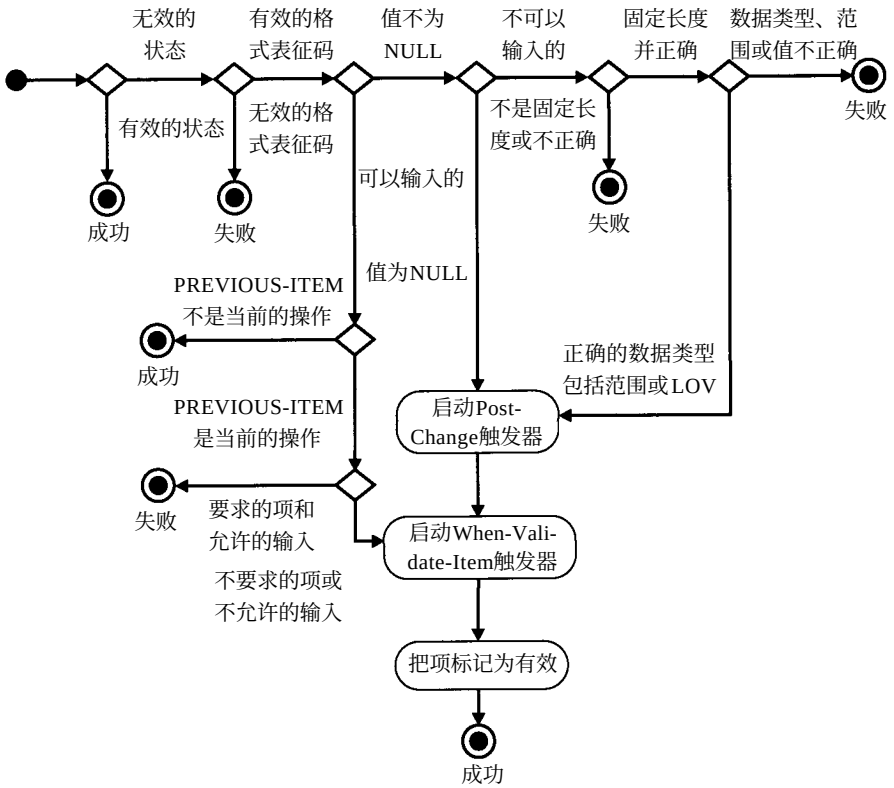

图5-4 项目的确认过程活动图

如在这个活动图中所看到的那样，确认主要是一系列关于各种有效性的判定。

可以用以下三种方法之一创建一个项目：

- Creating a record(创建记录)。创建记录中的项目作为New(新项)。Oracle Developer可以用缺省值填入或者从这个过程的其他项目拷贝它，但这个项目仍然是一个新的项目。
- Duplicating a record(复制记录)，把以前记录的值拷贝到一个新记录中，把新的记录设置为三个子状态中的任一个状态：新的、改变的或有效的。其状态是精确地复制以前

的那个记录项目的状态，即Oracle Developer所拷贝的那个记录的项目的状态。

- Fetching the record(取出记录)并立即使记录的项目有效，为每个项目执行 On-Fetch和Post-Change触发(后者是为了相容性而存在的，一般不应该使用它)。本章后面的“ 取出记录 ”将给出细节。

当用其中任何一种方法修改项目时， Oracle Developer把这个项目标记为Changed。项目的确认只对新的(New)和被改变的(Changed)项运行。当有一个New或Changed项目而且确认项过程的结果是True时，Oracle Developer启动Post-Changed和When-Validate-Item触发器，然后把这个项目标记为有效的 (Valid)。当确认项目的过程结果失败时，这个项目则保持原来的样子，而且游标继续停在这个项目上。

提示 不要使用Post-change触发器，因为Oracle要淘汰它，而应使用When-Validate-Item触发器。

项目由记录支配，所以清除个别项的唯一方法是清除记录。可以在项目的任何状态下进行清除。可以设置Clear-Item属性为项目的缺省值，但是，这并不能够从存贮器上把这个项目清除，只有除去记录才能把项目清除。

在确认一个项目时，该确认过程检查以下的属性，就像在图 5-4中看到的那样：

- Format mask:(格式掩码)：文本项目值的格式(可以是Valid(有效)或Invalid(无效))。
- Required and input allowed(要求和允许输入)：项值是否必须存在。
- Fixed lengh(固定长度)：项值是否必须是确定的那个长度。
- Data type(数据类型)：值的基本类型，比如数值型或日期型。
- Range(范围)：可能的最小值和最大值。
- List of values(值的列表)：值列表对象(LOV)的可能值的列表。
- Formula(公式)：如果域内有一个公式，那么这个公式是用于计算的。

如果确认了所有的内容，则这个进程为任何要附加确认的项目启动 Post-Change和When-Validate-Item触发器。如果触发器成功，就把这个项目标记为有效 (Valid)。

对于上面的项目确认属性的列表，可以看到在 When -Validate-Item触发器中只有很少的东西需要编码。假设你的数据库有 CHECK约束，它用简单的SQL表达式来确认在插入和更新中的个别项，所有这个内部检查都给定的话，会给 When-Validate-Item触发器留下什么？

一般地说，对复杂的多项确认，使用 When-Validate-Item触发器。通常，某种业务规则可能涉及多个项或者甚至涉及某种数据库查找。例如，在 Talbot农场的购买管理员为Buy(买)事务输入价格的操作中，会计部门对使用数据库价格的用户有一个限制： Authorized Purchase Limit(授权购买限制)，则可以在价格项目上对When-Validate-Item触发器编码，类似于这样：

```
DECLARE
  vPriceLimit NUMBER = 0;
BEGIN
  SELECT PriceLimit
    INTO :vPriceLimit
    FROM AuthorizedPurchaseLimit
   WHERE USER = AuthorizedUser;
  IF :Purchase.Price > vPriceLimit THEN
    Form_Trigger_Failure;
END;
```

When-Validate-Item触发器的另一个比较成熟的用法是用于为组合框的列表编程。组合框

是这样一个项目，它提供给用户一系列列表值，并且让用户利用 PL/SQL(这种技术超出了这本书的范围)输入不在列表中的值，在列表中添加新值时，可以使用 When-Validate-Item触发器。

(3) 记录的状态

记录有三种确认状态：

- New(新的)：在创建记录时，记录是新的。
- Changed(被改变的)：当改变了记录中的任何项时 (参看前面部分)，这个记录变成了被改变的。
- Valid (有效的)：在Oracle Developer确认了记录中的所有被改变的项或新的项并确认了这个记录后，或者在它从数据库中取出记录后，或者把记录提交到数据库以后，记录成为有效的。

复制记录即拷贝了源记录的状态。

记录的确认和登记只对被改变的记录进行。图 5-5显示了记录的确认过程活动图，有关登记过程的信息请参看早先的“ 登记数据库 ”的内容。

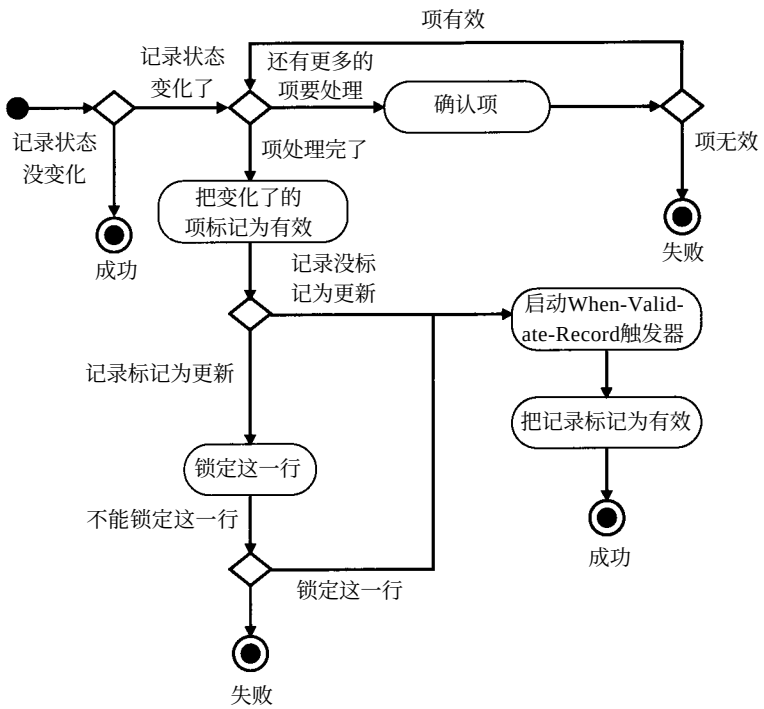

图5-5 记录的确认过程活动图

记录的确认进程要对记录中所有的项重复进行确认。如果其中的任何一项没能通过确认，那么用一个错误信息停止这个进程。在确认了所有的项以后，这个进程通过触发器把变化的项目标记为有效的，不必再重新确认它们。因此，如果记录被更新，那么进程锁定这一行 (有关细节请参看“ 锁定 ”)，然后启动 When-Validate-Record触发器，并标记这个记录为有效(Valid)。

When-Validate-Record触发器是一个逻辑位置，可以放置任何业务规则的检查，这些检查涉及一些不同的项目，这些项目在记录变成数据库的数据行时，其值必须一致。也可以利用

它来检查数据库或数据块其他一些项目的值。许多这样的约束和 When-Validate-Item触发器的情况一样，作为基础表上的 CKECK或FOREIGN关键字约束来表示，比作为 When-Validate-Record触发器的代码表示要好。另一方面，在某一个记录有某一方面的错误时，用户能立即得到反馈。触发器失败，显示信息会直接告诉用户在什么地方错了，而不是依靠关于约束失败的含义模糊的DBMS错误信息。

(4) 表单和块的确认

确认表单意味着确认表单上所有的块。在 Oracle Developer完成了块的确认以后，它检查在表单中是否带有触发器标记为 Change状态的任何项或记录，如果有，它把这些项或记录标记为Valid，则不用进一步确认它们。这个特性让你在确认触发器中做需要做的事情而不用无限循环。它可能对确认问题有影响，所以应该小心，使得这些触发器中的每件事情都正确 (见图5-6)。

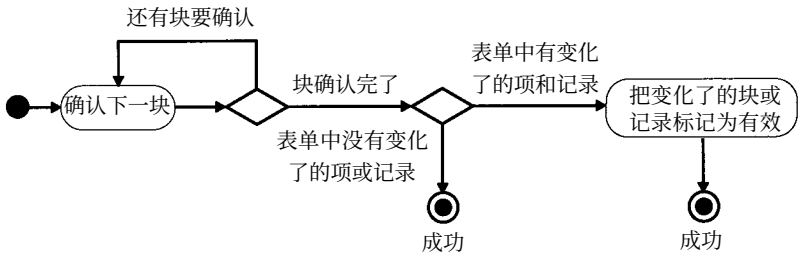

图5-6 表单和块的确认

确认块的过程随确认的单元而定稍有不同。确认单元可以是表单数据块或项。请参看前面“ 确认单元和状态 ”。如果单元是数据块或表单，那么 Oracle Developer确认数据块中的所有记录。如果确认单元是项或记录，它只确认当前的记录，如图 5-7所示。

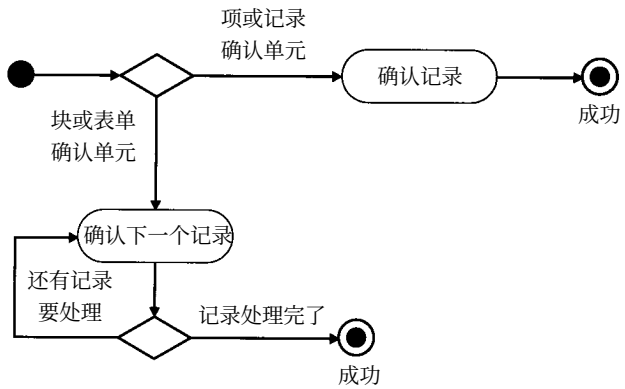

图5-7 确认当前的记录

5. 查询处理

查询是对数据库数据的请求。 Oracle Developer Forms应用程序有内部的查询进程，一行代码也不用写，可以提供巨大的能力和适应性。下面这部分帮助你了解查询进程不同部分的逻辑。图5-8概括了表单的查询处理过程。

创建查询意味着指定想加到数据库上的逻辑条件。在 Oracle Developer中，如果要这样做(就是说，输入一个查询)，就应在一个块上创建一个实例记录，然后输入某些条件，将条件附

加到缺省的查询上。这种指定查询的方法叫做实例查询 (query by example)，因为它是用提供想看的数据的实例来规定查询。这个查询还包含一组条件和缺省的排序说明。如果希望的话，可以通过专门的对话框输入更复杂的 SQL条件，然后执行这个查询，并取出记录，在块中显示它们。这部分讨论三个主要的进程，每个进程有一些子进程，其他进程可以共享这些子进程。

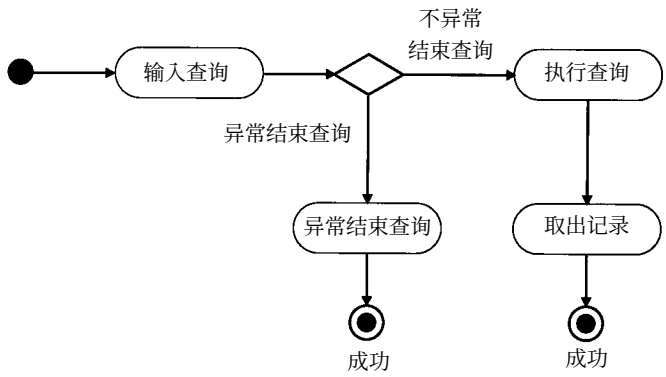

图5-8 表单的查询处理过程

还有两个进程：

- Count the query (查询计数)：这个进程为查询做准备，并要求服务器告诉你这个查询在指示信息中返回多少个记录。这个进程启动 On-Count触发器，有的话，替代计数进程。
- Abort and close the query (异常结束和关闭查询)：在输入查询时，你可能想不执行它就结束。这个进程启动On-Close触发器，如果有的话，它替代关闭进程。

1) 输入查询 对单个表查询的SQLSELECT语句有四个主要的子句：

```
  SELECT select-list
    FROM table
   WHERE query-condition
ORDER BY order-by-list
```

在Oracle Developer Forms应用程序中输入一个查询时，是通过实例建立这种类型的SELECT语句。 Selsct-list是一组对应于记录项的列项的名字。 Table是对应于块的基础表。order-by-list是select-list中的列项的列表，它规定被查询记录的次序，通过 ORDERBY子句块属性或通过Where对话框，可以交互式地规定order-by-list。

Query-condition (查询条件) 是一组SQL表达式，规定要返回哪些记录，这组表达式有三部分：

- Default conditions (缺省条件)：用来规定作为块属性的缺省 Where子句的一组条件，可以通过触发器在运行时改变这些条件。
- Column conditions (列条件)：通过填写实例记录的列所输入的条件。
- Special conditions (特殊的条件)：通过Where对话框输入的条件。

可以通过数据块的Query Data Source Type属性，将查询建立在程序、事务触发器或显示的FROM子句上。在前两个例子中， Oracle Developer并没有使用各种条件，而是为程序提供自变量或引用触发器的项。在最后一个例子中，有所有的条件，但是，可以引用连接的任何列。有关使用程序作为数据源类型的更多的细节，参看第 6章中的“过程的封装”部分的内

容。

当用户输入查询时，他或她可以使用关系操作符 (>、<、=、！=等等)和其他的SQL特性作为列的条件的一部分。这允许一些相当复杂的查询不违反“ 通过例子 ”的方法。如果需要，用户可以避开单独的对话而直接输入 WHERE子句的SQL代码。

在输入查询时， Oracle Developer并不对项或记录进行确认。请使用标准的关键字定义而不是执行关键字触发器，并禁止各种功能键 (主要是记录导航功能和提交处理功能)。

注意 如果你希望能够覆盖一个特殊的关键字的话，可以把关键字触发器的“ Fire in Enter-Query Mode ”属性设置为True。但可能会引起相当大的混乱，所以，在这样做时，要以常规模式和Enter Query模式测试表单。

图5-9输入查询的进程流图：

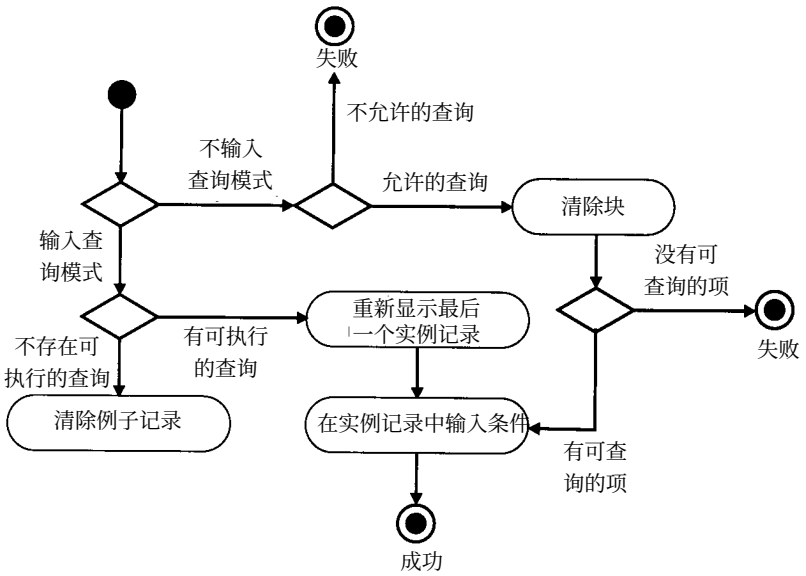

图5-9 输入查询的进程流图

Block属性表中的Query Allowed属性可以禁止对块的查询。可以通过设置 Database Data Block为NO规定一个块为control block(控制块) —— 没有基础表作基础的块。在块上至少应该有一项是来自数据库的基础表。

2) 执行查询 在输入查询条件以后，可以用菜单、工具条或为执行查询而创建的其他接口执行查询。当然，这并不是十分简单的。不首先输入查询也可以执行查询，这意味着你要利用缺省 WHERE子句或查询程序，取回所有的行。可以当块中有变化的记录时执行查询，Oracle Developer会问你是否提交变化。下面讨论与取出的行的数目有关的一些复杂关系。

图5-10展示了查询的执行情况。为了简化，活动图没有说明把游标放在第一个记录的过程(或者当查询不返回记录时返回一个错误的过程)。

在执行查询时， Oracle Developer首先检查这个块是否允许查询 (像在前面部分讨论的那样)。如果允许，它导航到这个块，确认块中那些没有被确认的记录。如果块中有 Changed的记录，那么它提示询问是否提交或清除。

Oracle Developer然后触发这个块的 Pre-Query触发器。这个触发器提供另一个地方指定

WHERE列条件。Oracle Developer现在检查这个块是否有基础表，虽然重复了对这个块的检查，但你可以不做前面的检查而从另外的地方进入这一点，比如从计算查询的行进入。如果检查成功，则Oracle Developer建立SELECT语句并启动Pre-Select触发器。如果有On-Select触发器，就执行该触发器，否则执行 SELECT语句。启动Post-Select和When-Clear-Block触发器并刷新块中的实例记录，然后取回记录。

注意 如果因为某种原因你指定的查询导致错误和很长的处理时间，而你在 Windows Registry中设置了CNTL_BREAK=ON的话，那么可以用CTRL-C键终止这个查询。

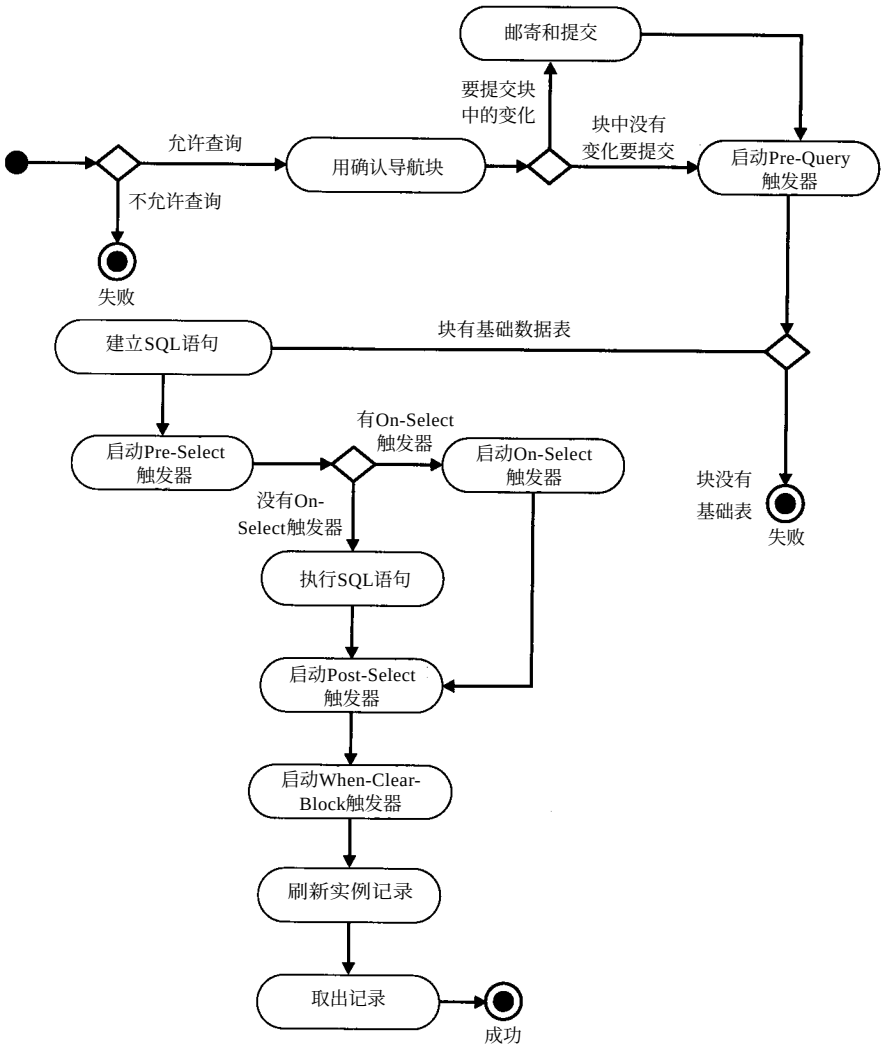

图5-10 展示执行查询进程的活动图

3) 提取记录 在执行SQL查询的过程完成以后，Oracle Developer就组织将记录提取出来。这是一个复杂的过程。它根据几个变量从数据库中提取数据行：

- 是否依赖数组处理。
- 查询从数据库中选择的数据行数。

■ 块中一次可见的数据行数。
■ 块在存贮器中存贮的行数。
■ 产生提取的操作(删除记录、滚动记录、或任何数量的其他可能的操作)。

如果有On-Fetch触发器，那么Oracle Developer启动这个触发器，而不从数据库中取一行放入记录缓冲器。

图5-11显示了提取数据的过程。首先检查已经取出放在缓冲器的等待行的列表。如果有这样的行，进程创建一个记录，并把列表中的当前行读入这个记录。如果没有等待的行，那么进程只是启动，或者从服务器取回更多的记录。如果当前的活动是 Clear Record (清除记录)，那就意味着需要取出另外一行填入这个空间。否则，要按照你在块的 Query Array Size属性里规定的数目来提取数据行。进程用适当的值设置块的 Record to Fetch属性。如果有On-Fetch触发器，则进程执行它，而且可以按照在块属性 Record to Fetch中规定的提取记录数去用 Get_Block_Propery访问那些记录。否则，进程提取出所要求的数目的记录并创建一个记录，把新的等待列表中的第一行读入这个记录。

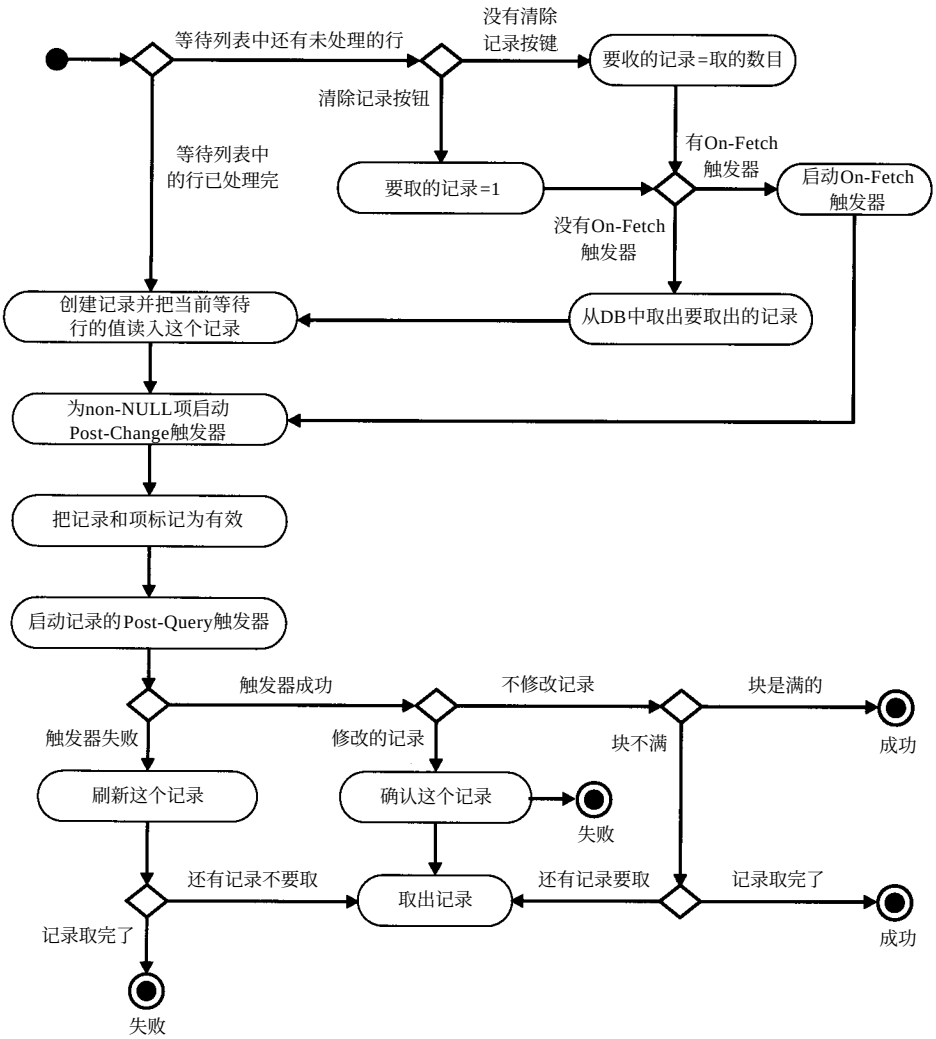

图5-11 提取记录的活动图

在记录创建完成之后， Oracle Developer循环通过新记录中的非空项，执行所有的 Post-Change触发器。因为在这些触发器中有对处理的限制，而且它的存在只是为了向后兼容，所以应该避免使用它们，而只须依靠 Post-Query触发器来完成取出记录的任务，比如确认或传送数据到记录的其他项中。在所有的项处理完后，进程把所有的项和记录标记为有效。

在把项标记为有效以后，进程为这个记录执行 Post-Query触发器。例如，利用这个触发器去设置那些其值与基础表中的列的值不对应的项，或者用它确认项和记录的条件。如果 Post-Query触发器失败，那么进程刷新这个记录，意味着它将不在显示的记录中出现。当发生这种情况时，进程重新启动自己，开始提取下一行。

注意 Developer并不通知用户它刷新了记录而且不显示它们，它只是“默默地”去做这件事，这可能会引起混淆。例如，如果把确认代码放入Post-Query触发器中，而数据库中50个记录中有10个记录在确认时失败，那么用户只能看到触发器成功的40个记录。这就意味着用户不能访问那些更新的记录。如果这正是你希望的，那很好。否则，除了在这个触发器确认失败时缺省调用Form_Trigger_Failure以外，还要做某些事情，例如，可能要显示错误警报或在表中记录这个错误。

Post-Query触发器的主要用法是用从数据库表中查找的值去填写非数据库项。例如，在人员表中，人员的地址信息由嵌套对象类型的若干列组成。你可能希望按照信封地址的约定在连续的和格式化的单个域内显示这个信息，而不是单独地显示所有这些列。如果街道地址只有一行的话，你也可能希望取消 Street2(第二行街道地址)。你把个别的值取到项中，但是却不能个别地显示它们。取而代之是使用如下的代码，在 Lodging 块的Post-Query触发器中填充数据库的非数据库文本项，然后显示地址：

```
:Lodging.DisplayAddress := :Lodging.Street1 || CHR(10);
IF (:Person.Street2 IS NOT NULL) THEN - only show Street2 if it isn't null
  :Lodging.DisplayAddress := :Lodging.DisplayAddress || :Lodging.Street2 || CHR(10);
END IF;
:Lodging.DisplayAddress := :Lodging.DisplayAddress || :Lodging.City || ", " ||
  :Lodging.State || "  " || :Lodging.PostalCode;
```

提示 ASCII代码10是回车，它可以打断地址部分之间的行。CHR函数是一个Oracle函数，它返回ASCII字符的整数值。

如果Post-Query触发器成功，那么进程根据触发器是否修改了记录来决定做什么。如果记录被修改，则进程要对这个记录进行确认。如果确认失败，则用一个错误信息终止提取进程，否则，进程继续提取更多的记录。

如果块是满的，意味着 Number of Records Buffered(缓冲的记录数目)属性设置的存贮器不能保存更多的记录，则当前的提取停止。如果块不是满的，而且要提取有更多的记录，那么进程从开始到提取下一个记录循环。

注意 实际上很少超过存贮器的限制，除非你把Number of Records Buffered(缓冲记录的数目)设置为(除了)缺省外的某个值。应该不干涉这个属性，除非愿意承担为了证实你的方法所要求的彻底的测试。

6. 导航的处理

当用户到处移动应用程序时，有几个触发器可以起到特殊的作用。触发器启动的进程包含有创建缺省值的各种方法，比如，在第一次输入一个块时，创建缺省的新记录。

进入对象，则启动该对象的触发器有：Pre-Form、Pre-Block、Pre-Record、Pre-Text-Item，离开对象时启动的触发器有：Post-From、Post-Block、Post-Record、 Post-Text-Item。

移动到不同的对象时启动的触发器有： When-New-<Obiect>-Instance、When-New-Form-Instance、When-New-Block-Instance、When-New-Record-Instance、When-New-Item-Instance。这些触发器在要求某种初始化的情况下，非常有用。例如在鼠标环境下对实现有条件的导航就非常有帮助，当表单用具体示例说明对象时，它们对定位到某处特别有用 —— 比如要定位到叠状画布的项或对话框的项。这是因为这些触发器允许使用带限制的内部子程序比如Go_Item和Go_Block，而导航触发器不允许这样。在本章后面有这方面的更详细的内容。

在启动一个新表单时， Oracle Developer启动When-New-Form-Instance触发器。具体地说，当导航到表单的第一个可导航的块的第一个可导航的项以后，那个触发器被启动。这就是意味着所有的导航触发器，包括 Post-Form、Post-Block、Pre-Form、Pre-Block、Pre-Text-Item等等，在When-New-Form-Instance触发器以前启动， When-New-Form-Instance触发器在把控制返回给用户之前启动，即正好让用户输入数据到表单以前启动。

顶层表单的When-New-Form-Instance触发器的共同的任务是控制 MDI窗口的大小。例如下面的When-New-Form-Instance触发器代码将窗口设置为 600×300个像素(如果Coordinate System是像素的话)：

```
Set_Window_Property(Forms_MDI_Window, WINDOW_SIZE, 600, 300);
```

当导航到不同块的项时， Oracle Developer启动When-New-Block-Instance触发器，启动它是在导航触发器之后和在把控制交给用户输入以前。可以利用 When-New-Block-Instance触发器来初始化内部的块元素，比如那些数据库不自动填充的列表和记录组。也可以在确认块以后，利用它控制导航，因为离开一个块通常意味着进入另一块。

当移动到不同的记录上时， Oracle Developer启动When-New-Record-Instance触发器。每当把焦点移到一个块中的记录并希望发生某件事情时，可以利用这个块的这个触发器。触发器名字中的“ New ”(“ 新 ”)在这里是“ 不同 ”的意思。每当移动到记录时而且不管移入它多少次， Oracle Developer均启动这个触发器。这个触发器在任何导航的触发器之后但在把控制返回给用户输入之前启动。可以利用 When-New-Record-Instance来初始化数据块记录的内部元素，或者可以在前面的记录被确认之后利用它导航。 When-New-Record-Instance的主要用法是在记录确认以后导航到新表单或另外一个块或项。因为 When-Validate-Record不允许使用限制的内部程序，比如Go-Record，所以必须依靠When-New-Record-Instance来控制导航。例如，可以在 When-Validate-Record触发器中设置一个控制项或全局项，然后在 When-New-Record-Instance触发器中检查它，导航并重新设置这个项。

在得到用户的输入以前，移到一个作为最后目标的不同项时， Oracle Developer启动 When-New-Item-Instance触发器。当输入焦点移到某个项上时，如果想激活触发器，可以使用这个项上的这个触发器。在导航到达某个项并停在这个项得到用户的输入以前时，这个触发器并不启动。当移到报警域或菜单项时它也不启动。你将发现这个触发器对在导航完成以后调用受限制的内部程序特别有用，它对表单的特性有非常好的控制。

7. 注册处理

Oracle Developer Forms应用程序的注册和注销要在几个地方激活触发器。注册进入一个表单应用程序中，要设置各种选项、连接数据库、为当前表单的所有块设置列的安全性。图 5-12

是注册进程的活动图。进程检查是否已经注册，如果没有注册，则加以处理。在启动 Pre-Logon触发器以后，进程检查连接是否有效。如果没有连接，并且存在 On-Logon触发器的话，它就启动这个触发器。如果没有这种触发器，它将尝试连接到服务器上。在三次努力均不成功之后就失败。

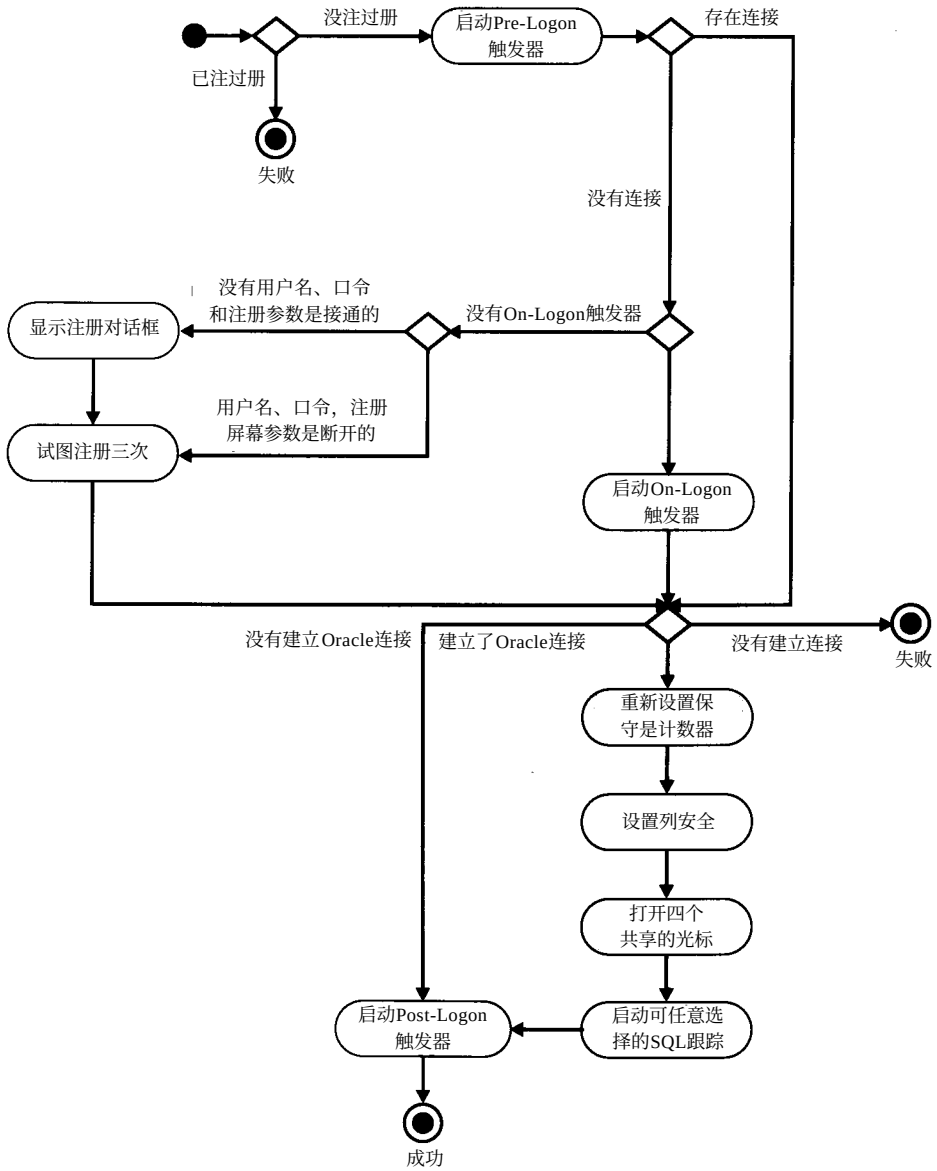

图5-12 注册活动图

如果要连接到Oracle数据库服务器上，和某些其他的数据库管理程序相反， Oracle Developer建立Oracle专用客户方的对象(保存点、列安全性、共享游标、 SQL跟踪)。Oracle Client Adapter (Oracle用户适配器) 使得第三方的数据库管理程序可以利用这些特性。有关详细信息请参看第14章的有关内容。

在重新设置保存点计数器之后，注册进程通过表单中所有的块，检查各个列的安全性约

束。在想要约束的块中，设置Columu Security(列安全)属性为True，使得可以对它进行安全性检查。在任何这样的块中，操作员必须在服务器的数据词典中有列层的更新特权，或者注册进程把这个项的Update Allowed属性设置为off。可以通过定义一个On-Colunu-Security触发器定义安全进程。

提示 Oracle公司建议不要使用这些属性，因为使用这些结果是使你的模块更加复杂。

实施了安全性检查以后，进程打开四个的游标，如果需要的话便启动跟踪，并启动 Post-Logon触发器。

注销一个表单应用程序比注册简单得多。即注销清理与数据库有关的存贮器并注销数据库。首先，进程启动Pre-Logout触发器，然后关闭Oracle游标。如果有On-Logout触发器，那么进程执行这个触发器，而不是注销服务器和启动 Post-Logout触发器。

客户机/服务器应用程序的一个共同任务中把这个应用程序与不同的用户名相连接。当没有用用户名、口令或连接字符串启动Oracle Developer应用程序时，Oracle Developer显示初始表单并提示需要这些信息。

连接成功后，可能希望允许注销并连接到另外不同的用户名上，缺省菜单或表单中的有效活动集没有这部分功能。可以在工具条按钮项上用类似于以下的编码写一个象 When-Button-Pressed那样的触发器：

```
DECLARE
   vUser     varchar2(80);
   vPassword varchar2(80);
   vConnect  varchar2(80);
BEGIN
   Logout;
   Logon_Screen;
   vUser := Get_Application_Property(USERNAME);
   vPassword := Get_Application_Property(PASSWORD);
   vConnect := Get_Application_Property(CONNECT_STRING);
   IF vConnect IS NOT NULL THEN
      Logon(vUser, vPassword||'@'||vConnect);
   ELSE
      Logon(vUser, vPassword);
   END IF;
END;
```

当用户在这个按钮上单击时，这个触发器显示标准的 Oracle Developer注册屏幕，在屏幕中输入新的用户名、口令和连接字串，并按该用户名为用户注册。如果想在输入的信息中进行错误检查的话，可以用自己设计的对话框替换标准的注册屏幕，

注意 在试图用另一个用户名注册之前，你必须先注销，否则会因为重复注册而失败。这意味着如果用户提供了一个错误的用户名、口令、或连接字串，应用程序将不连接到数据库。应该通过某种警告或信息，确保让用户了解这个情况，就象On-Error触发器那样。(有关细节请参看“信息处理”部分的内容)。

8. 数据块和记录的处理

数据块是表单中所发生的许多事件的中心。数据块是一个对象，它是一组项目的源。大多数数据块映射到服务器上的单个基础表上。但是也可以有控制块，这些块不映射到数据库表。利用控制块把不是来自数据库的项目加到表单中，形成不适合连接到数据库的块。在任

何情况下，数据块都是包括了项目的这种基本结构，而且它提供了它所包含的项目组的聚集层。可以利用这个层次完成许多不同的动作和作用。

除了前面讨论的广泛的查询手段以外，有一些基本的数据块进程启动应用程序的主要部分。

虽然创建记录很简单，但它是一个主要的进程。图 5-13展示了这个进程的活动过程。

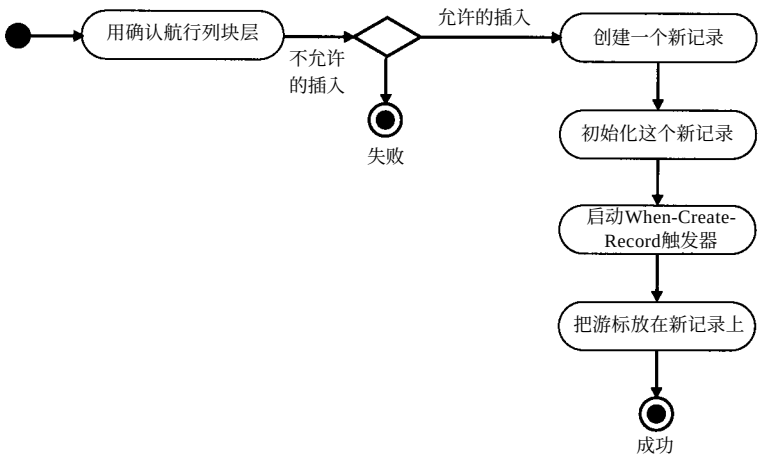

图5-13 创建记录活动图

在导航到带有确认的块的层次时，进程创建一个记录并初始化这个新记录（如果这个数据块允许创建新记录的话），然后进程启动When-Create-Record触发器，并把游标放在新记录上。可以用各种方式为改变了的项和记录做标记。有关的细节参看前面的“确认”部分的内容。对于Oracle Developer应用程序的事务处理，这个进程是重要的。这个进程的活动见图 5-14。

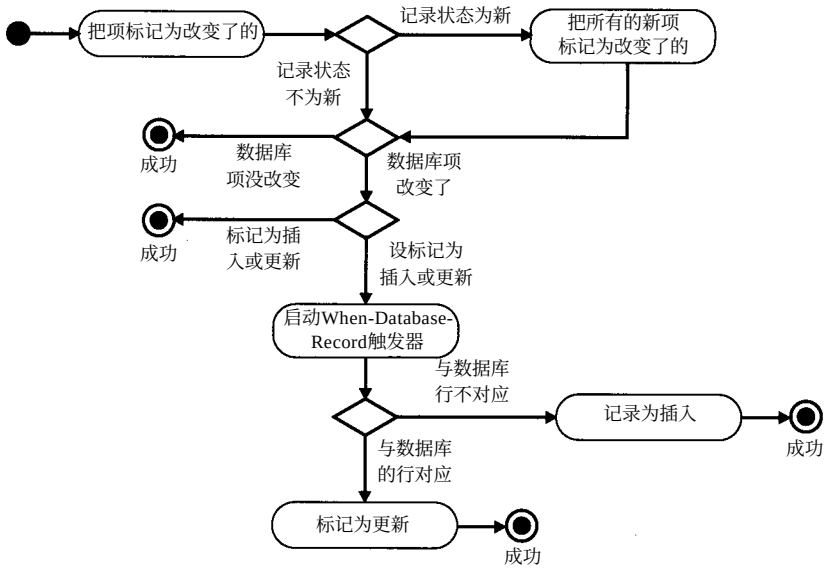

图5-14 进程活动图

进程首先将项目标记为“被改变的”。如果记录是“新的”，那么进程把这个记录的所有项都标记为“被改变的”。检查那些标记为“被改变的”数据库项，如果有这样的数据库项，

而且Oracle还没有把这个记录标记为插入或更新，那么进程启动When-Database-Record触发器，然后把数据库行(通过查询来自数据库的行)标记为Update(更新)而把其他的行标记为Insert (插入)。

清除数据块时先确认这个块，并提示是否要提交。如果希望数据块登记入库并且要提交，则启动When-Clear-Block触发器，刷新当前记录，并把游标放在当前块上。清除记录则有比较复杂的逻辑关系，要重新排列块和显示中保留的记录。在这个进程中启动 When-Remove-Record触发器是最重要的事件。

9. 接口和按键事件的处理

在Oracle Developer Forms应用程序中有几十个接口事件。庆幸的是，这些事件的有关逻辑和它们在应用程序中的处理不很复杂。通常，事情的发生是引发事件 (单击鼠标、按下按钮、检查复选框、关闭窗口等等)，以及由与这个事件有关的触发器触发引起。

利用这些触发器，可以为表单应用程序的大多数接口事件定义所发生的事情：

- 用户接口控制：可以为块中的不同项，比如，图像按钮、复选框、单选按钮和列表，定义所发生的事情。
- 鼠标事件：可以定义在鼠标单击、鼠标双击、鼠标下移(mouse-down)、鼠标上移(mouse-up)等操作后做些什么。
- 按键事件：可以定义各种功能键：Clear Block、Insert Record、Previous Block、Enter Query (清除块，插入记录，前一块，输入查询)等，定义键盘功能键(F0-F9)的功能。

注意 用Key-Others触发器去取代任何功能键时，功能键可能会有触发器，但在任何的表单层都没有定义这种触发器。可以利用这个触发器关闭所有键的功能，例如当用户处于对话框中时就可以这样作。

10. 主-从协调

Coordination-causing(协调原因)事件是主-从关系(master-detail pair)中的主块的某个事件，它使主块的不同记录成为当前记录。

Oracle Developer处理所有事件的实际过程是通过某些缺省产生的触发器来进行的，这些触发器是：On-Clear-Details、On-Populate-Details、On-Check-Delete-Master。Oracle Developer堆积这些触发器，并在下一个事件以前把它们当作堆放的一连串进程的一部分来处理。这个特性与表单应用程序其余的逻辑相比有点不一样。如果打算修改这些触发器，由自己来做协调工作，要准备进行逻辑实验，直到完全了解它为止。

当定义主-从关系时，表单创建On-Clear-Details触发器。当引起协调事件发生时，如果这个关系的Auto-Query属性是False，那么将自动清除从属块。

当定义主-从关系时，表单也创建 On-Populate-Details触发器。当引起协调事件发生时，它自动填充从属块。

当在关系属性表中把 Master Deletes属性设置为Non-Isolated时，表单创建 On-Check-Delete-Master触发器。这意味着如果有和主记录联系的从记录的话，要能够防止删除这个主记录。

还有一些逻辑必须设置Auto-Query，它允许延迟填充从属部分直到导航进入从属块为止。

提示 在重新安排表单数据块时，可能遇到这些触发器不一致的问题，可以利用 Data

Block Wizard通过删除和重新创建关系在任何点重新产生它们。也可以不用向导而通过它的属性表来创建关系对象并直接输入连接条件。如果在数据块模式框架外工作，则必须人工做这个工作，因为没有框架的块不能再使用Data Block Wizard。

11. 信息处理

在Oracle Developer Forms应用程序中有两个触发器影响信息的处理： On-Error和On-Message。前者是在Oracle Developer试图显示错误信息时启动，而后者是在Oracle Developer试图显示指示信息时启动。可以利用这些触发器来处理错误和信息。在触发器编码中，可以访问内部函数，从而返回各个部分的信息：

- Error_Code 或Message_Code ：返回错误编号。
- Error_Text或Message_Text：返回文本信息。
- Error_Type或Message_Type：返回信息的种类。
- DBMS_ERROR_Code：返回服务器错误编号，它可能和表单的错误编号不同。
- DBMS_Error_Text：返回服务器信息，它可能和表单的错误信息不同。

如果需要，可以抽取信息的任何元素并在进一步的处理中使用它，或者提供你自己的错误处理。通常，这样做的方法是在表单层对 On-Error触发器和On-Message触发器进行编码。这些触发器调用标准库的程序来格式化和显示信息。例如，下面是一个 On-Message触发器，它取代标准的表单信息处理：

```
IF Message_Type = 'FRM' THEN
  Handle_Forms_Message(Message_Code, Message_Text);
ELSE
  Handle_Database_Message(DBMS_Error_Code, DBMS_Error_Text);
END IF;
```

这个触发器对数据库信息和表单信息加以区别，并调用标准库中带有适当的信息代码和文本的子程序，这些子程序将使用标准的格式和专门的信息用于你的应用程序中。例如，如果在打开另一个表单以前不想显示来自登记变化的信息“ Database apply complete：<n>records applied OK ”，可以创建如下所示的Handle_Forms_Message程序：

```
PROCEDURE Handle_Forms_Message(pCode IN NUMBER, pText IN VARCHAR2) IS
BEGIN
  IF pCode = 40404 OR pCode = 40405 THEN
    null; -- do nothing for this message
  ELSE
    Message('Forms '||To_Char(pCode)||':  '||pText);  -- format message
    Message('');  -- stack message to display alert, clear status bar
    Synchronize;  -- display message
  END IF;
END Handle_Forms_Message;
```

这个程序忽略40404号和40405号信息，利用显示警报而不显示信息条上的信息的编程技巧，显示任何的其他信息。如果信息堆栈中有多个信息要显示，那么 Forms用第一条信息弹出一个警报，然后在状态条上显示最后的信息。这个程序利用这个特点使表单显示警报中的所有信息。有一些事件你可能想做或者可能不想做，这取决于应用程序的编程形式，内部同步机制告诉Forms采用信息堆栈同步显示，就是说，现在立即显示信息。

有三种Forms信息：工作信息，指示信息和错误信息。有关特殊信息的资料，请查询Forms联机帮助。可以用下面一行PL/SQL代码取消工作信息。

```
:SYSTEM.Suppress_Working := 'TRUE';
```

可以用ON-Message触发器处理指示性信息，在 Message_Code，Message_Text和Message_Type中有这些信息的资料。可以用 On-Error触发器处理错误信息。在 Error_Code、Error_Text和Error_Type中有关于错误的资料。显示信息的细微控制来自 System.Message_Level变量的使用，这个信息的行值表示Forms信息各种不同的严格程度，如表5-1所示。

表5-1 Forms信息的严格级别

严格程度	说 明	信息实例
0	任何信息	
5	明显的条件	在第一个记录上
10	程序的错误(要做的事情超出了命令的范围)	域是满的。不能插入字符
15	设计错误(要做的事情不能通过设计在表单里做)	试图更新一个不允许更新的域
20	触发器失败或阻止用户前进的条件	不能在视图中插入或更新数据
25	阻止表单正确执行的条件	由于以前的回滚，更新不能进行。清除一个记录
>25	Forms始终显示的错误	缺乏存贮器

这个变量的值确定了Forms显示什么信息。表单只显示所设置的以上等级的信息。例如，如果把变量设置为‘5’，那么Forms就禁止严重程度为0或为5的信息。利用这个变量，可以关闭那些被认为只会把用户弄糊涂的警告信息。

```
:System.Message_Level := '5';
```

每当要使用适当的触发器时，可以接通和关闭信息。可以查阅联机帮助，找出各种信息的严重程度。

警告 不要将一连串的操作的Message_Level设置为大于零。要尝试限制这个变化的范围，只需在可能发生混淆信息的短时间内消除混淆信息。如果长时间关闭信息，将发现调试和确定发生的问题非常困难，因为缺乏信息，你不可能知道真正的原因。

12. Restricted Built-in(受限制的内部子程序)

Oracle Developer的自动导航能力使得完成编程任务非常容易。然而另一方面， Developer复杂的导航和事件驱动处理具有潜在的无限循环。当 Oracle Developer Forms正在导航时(如果能够导航)，有可能陷入循环 —— 你导航，导致表单导航，这又引起你导航，这样不断继续下去。庆幸的是， Oracle Developer认识到了这种可能性，它使用有限的内部子程序使得很难出现这种情况。乍一看，这个特性似乎可能使编程更困难，但是真实的情况正好相反。

Restricted Built-in是在导航时开始执行的内部子程序。 Oracle Developer规定在响应导航启动触发器时使用Restricted Built-in是非法的。在实际中，这种限制通常在触发器中引起运行时的错误，它使你要使劲地去考虑在这个 Oracle Developer的“限制”周围发生了什么并如何去工作。稍为思考一下，去熟悉这些限制，这样可能对避免这些问题有所帮助。

代替在导航触发器中调用被限制的内部子程序的是，把调用放在由导航触发器单独触发的触发器中。例如，不应在 Post-Query触发器中调用Go_Item来将游标移到不同的项，而应该把Go_Item调用放在这个块的Key-EntQry触发器内。如以下所示：

```
Enter_Query;
Go_Item ('Ledger.ActionDate');
```

这个触发器由用户输入查询时触发，而不是由缺省的查询输入动作触发。在用户执行这

个查询和Forms提取了所有的行之后，控制返回到这个触发器，这个触发器再把它转送到Ledge.ActionDate项。因为只利用按键也可以进行类似的动作，所以还需要在这个块上定义Key-ExeQry触发器。

```
Execute_Query;
IF :SYSTEM.MODE = 'NORMAL' THEN
Go_Item ('Ledger.ActionDate');
END IF;
```

如果你也想这么做，最好的开始位置是用一组 New-Instance触发器(When-New-Form-Instance、When-New-Block-Instance、When-New-Record-Instance和When-New-Item-Instance)。彻底了解这些触发器的用法，可以节省相当多的时间。

例如，比方说，有一个确认处理的问题，根据当前记录的状态要求导航到一项或另外一项，做这种确认处理的地方明显是这个项上的 When-Validate-Item触发器(Ledger.Action)。当确认发生时，必须把焦点移到另外一项，但是 When-Validate-Item触发器不让调用Go_Item！

为了处理这种情况，需要使用“全局变量”的编程形式。建立一个控制块 (数据块的Database Data Block属性设置为NO)和一个文本项，可以用来当作是被调用的 vDestination的变量，调整它的尺寸使得可以容纳任何项的名字。通过利用以下语句，把对 Go-Item的调用放在Ledger.Action的When-Validate-Item触发器中：

```
:Control.vDestination := 'ActionType.Action';
```

这个变量将包含对话框的画布上的项的名字 (在这种情况下是ActionType.Action)。在这个块中，或者在按照顺序排列的下一个项的 When-New-Item-Instance触发器中，有以下代码：

```
IF :Control.vDestination IS NOT NULL THEN
  Go_Item(:Control.vDestination);
  :Control.vDestination := '';
END IF;
```

当用户移动出Ledger.Action项时，When-Validate-Item触发器启动，设置vDestination变量。然后这个触发器成功地处理这种情况，完成导航并把焦点移到下一项。在那里，启动 When-New-Item-Instance触发器。这个触发器到达被命名的 ActionType.Action项，然后重新设置变量。这后一个步骤确保能够继续正确处理进一步的导航，否则，当移动到 Ledger块的下一项时，将只是弹回到对话框。

这种逻辑允许某些富有想象的策略。如果 Oracle Developer能够“思考”的话，这里给出的例子就是对Oracle Developer“思考方法”的简单研究。不要企图使系统屈从你的意愿，应该了解它的进程是如何工作的并顺着它走。

提示 另外一种没有实际导航的导航方法：创建一个项作为指示标志，在项或表单层创建一个When-New-Item-Instance触发器，然后检查标志并导航到指定的项。当执行确认触发器时，设置这个标志，导航进行。当焦点移动到一个新项时， When-New-Item-Instance触发器启动，检查标志，并导航到指定的项。

警告 一个不做工作的“解决办法”是利用 When-Timer-Expired触发器。可以设置这个触发器在很短的时间内进行，使得它正好在导航停止后启动。这个触发器常出现在Oracle Developer的各个部分，因为在导航中触发器是不会触发的。计时器触发器在导航停止后几乎立即触发，然而，几乎还不够。任何熟悉多线程系统的人将告诉你，这

是一个很糟的想法。如果系统忙，不管时间间隔多么短，别的触发器可以先启动。如果是在web上使用，这更是个问题。因为每个计时器事件要求通过网络到达Application Server。所以应该使用进程逻辑，而不要使用技巧去解决这个特别的编程问题。

表5-2、表5-3和表5-4分别列出了不能使用被限制的内部子程序的触发器，可以使用被限制的内部子程序的触发器和一组被限制的内部子程序。

表5-2 不能使用被限制的内部子程序的触发器

On-Check-Delete-Master	Post-Block	Pre-Insert
On-Check-Unique-Trigger	Post-Change	Pre-Logon
On-Close	Post-Database-Commit	Pre-Logout
On-Column-Security	Post-Delete	Pre-Query
On-Commit	Post-Form	Pre-Record
On-Count	Post-Forms-Commit	Pre-Select
On-Delete	Post-Insert	Pre-Text-Item
On-Error	Post-Logon	Pre-Update
On-Fetch	Post-Logout	When-Clear-Block
On-Insert	Post-Query	When-Create-Record
On-Lock	Post-Record	When-Custom-Item-Event
On-Logon	Post-Select	When-Database-Record
On-Logout	Post-Text-Item	When-Image-Activated
On-Message	Post-Update	When-Remove-Record
On-Savepoint	Pre-Block	When-Validate-Item
On-Select	Pre-Commit	When-Validate-Record
On-Sequence-Number	Pre-Delete	
On-Update	Pre-Form	

表5-3 可以使用被限制的内部子程序的触发器

Key-<Whatever>	When-List-Changed	When-New-Item-Instance
Key-Fn	When-Mouse-Click	When-New-Record-Instance
Key-Others	When-Mouse-DoubleClick	When-Radio-Changed
On-Clear-Details	When-Mouse-Down	When-Timer-Expired
On-Populate-Details	When-Mouse-Enter	When-Window-Activated
When-Button-Pressed	When-Mouse-Leave	When-Window-Closed
When-Checkbox-Changed	When-Mouse-Move	When-Window-Deactivated
When-Form-Navigate	When-Mouse-Up	When-Window-Resized
When-Image-Pressed	When-New-Block-Instance	
When-List-Activated	When-New-Form-Instance	

表5-4 受限制的内部子程序

Block_Menu	Delete_Region	Help	Post
Call-Input	Down	Insert_Record	Previous_Block
Clear-Block	Do_Key	Last_Record	Previous_Form
Clear-EOL	Duplicate_Item	List_Values	Previous_Item
Clear_Form	Duplicate_Record	Main_Menu	Previous_Menu
Clear_Item	Edit_Text_Item	Menu_Next_Field	Previous_Menu_Item
Clear_Message	Enter	New_Form	Previous_Record
Clear_Record	Enter_Query	Next_Block	Scroll_Down
Close_Form	Execute_Query	Next_Form	Scroll_Up
Commit_Form	Execute_Trigger	Next_Item	Select_All
Convert_Other_Value	Exit_Form	Next_Key	Select_Records
Copy_Region	First_Record	Next_Menu_Item	Show-LOV
Count_Query	Go_Block	Next_Record	Terminate
Create_Queried_Record	Go_Form	Next_Set	Up
Create_Record	Go_Item	Open_Form	Update_Record
Cut_Region	Go_Record	Paste_Region	

5.1.2 报表进程和触发器

Oracle Developer Reports进程比前面讨论的表单进程要简单得多。用报表做的大部分工作，特别是重复循环，就是用数据模型组、重复的模式、报表类型这三部分的组合来进行说明。有关报表的创建和编程的细节，请参看第 7章的内容。有一些特定的操作，特别是涉及条件判断的操作，必须在报表触发器中采用定制方法进行程序编制。图 5-15展示了报表进程，它包含有几个触发器的触发点。 Before和After触发器的主要用途是创建一个报表使用的处理表。例如，可以用SQL语句或PL/SQL代码创建一个临时表，然后用这个表的值作报表，这样作的一个例子是资金平衡表报表。资金平衡表报表建立一个平衡表，并从初始的平衡表和事务表中汇总会计信息。

报表进程使用给定的报表运行系统的命令行参数启动。如果用户提供了某些参数，则Oracle Developer启动输入的确认触发器，这个触发器是通过参数属性工作表检查和 /或改变报表实例的参数来触发的。运行时参数表单让用户为报表输入运行时选项。 Oracle Developer启动Before Form和After Form触发器，以及可能再次启动确认触发器。

设置参数以后， Oracle Developer编译这个报表并启动Before Report触发器。执行报表的逻辑比图5-15显示的要复杂。用户可以介入的地方是在和报表上的任何对象联系的 Format触发器上，以及在打印完每一页 (最后一页除外)以后启动的Between Page触发器上。

打印完最后一页后，报表提交结束这个事务，然后启动 After Report触发器。

可以在Previwer(预览器)上运行报表，也可以只把结果发送目的地。如果选择预演报表，则可以交互式访问报表。可以利用某些触发器来完成交互操作，特别是利用 Action 触发器(在“作用触发器”部分有更多的内容)，可以通过这种方式把报表当作交互应用程序来处理。应该知道，在预演器中使用类似于‘向前引用’那样的操作可能会引起触发器以不可预料的方式启动。

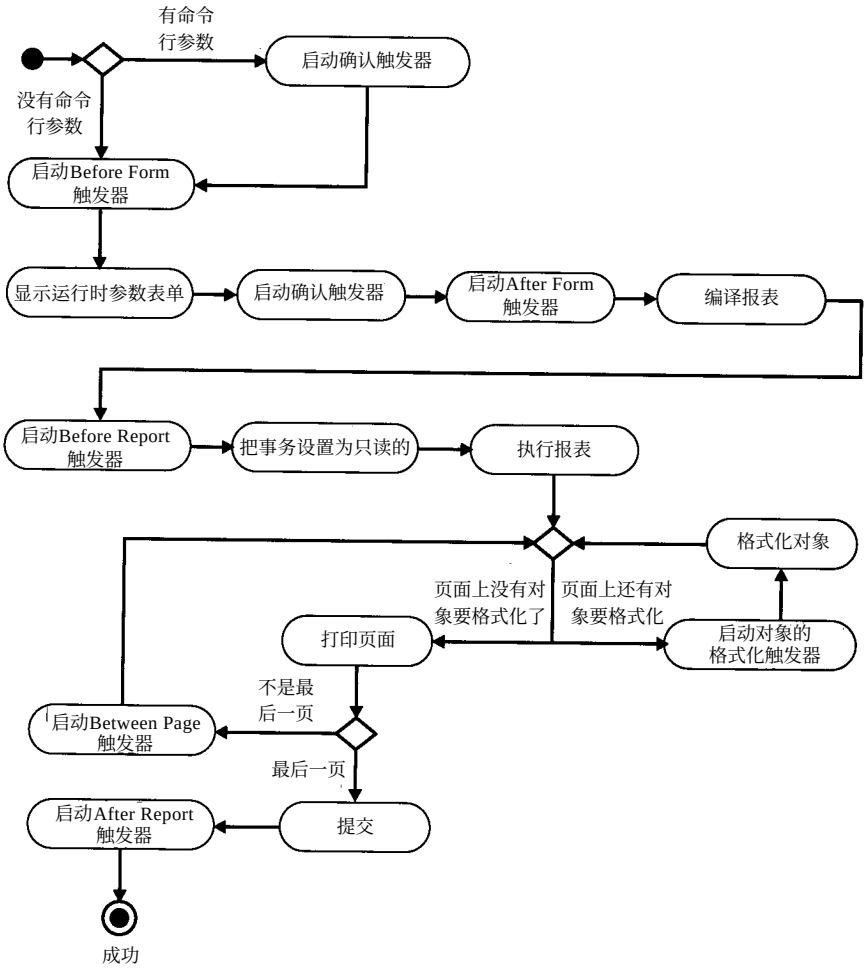

图5-15 报表进程的活动图

1. Report触发器

Report触发器(Before and After Report、Between Pages、Before and After Form)让用户在报表层和页面层上对报表处理进行程序上的访问。

Before Report触发器的触发是在报表对象的任何格式化之前和在为报表查询数据之后。可以用这个触发器进行初始化设置。如果这个触发器失败，那么 Oracle Developer显示一个错误信息并返回。

After Report触发器在打印完报表或退出预览器以后启动。可以在运行报表以后利用这个触发器来清除那些不希望留下来的任何东西。即使在报表中触发器失败，这个触发器也总是启动。应该明显地显示某些错误信息，因为当这个触发器失败时，它不做任何事情。

可以利用Between Page触发器定制页面的格式。在Live Previewer中，Oracle Developer只在第一次显示报表页时才启动这个触发器。如果这个触发器失败，那么 Oracle Developer停止格式化页面并显示一个错误信息。

Before Form和After Form触发器用来操作报表的参数。不能从这些触发器中操作数据模型对象，比如列。不管是否取消运行时参数表单， After Form触发器总要启动。如果这些触发器失败，那么Oracle Developer停止执行这个报表。

注意 在Before Report、After Report、Between Page触发器中应避免使用DDL和DML语句。当Oracle Developer在开始运行建立报表时，它提取表的快照，如果执行 DDL和DML语句，这个快照将不再反映正确的数据词典，而且这可能会引起难以理解的运行时错误。

2. Format(格式化)触发器

当格式化一个实例对象时， Oracle Developer调用在报表中与该对象联系的Format触发器。可以改变对象的边界、和格式化掩码，可以格式化那些通过 SRWPL/SQL库程序包访问的对象的任何其他方面。可以跳过在基于变量值的对象内的处理。

如前面所提起的注意那样，在这些触发器中，应该避免 DML和DDL语句，否则这些触发器的启动可能是不可预测的。在单个对象的格式化期间，这些触发器可以启动许多次。

通过在域的周围放一个框架对象，而且把 Format触发器和框架联系起来，可以格式化矩阵中的单元。

3. Action(作用)触发器

在Previewer中运行报表时，为了能够做一些交互操作，可以在报表上放一个按钮。当这个按钮被按下时， Action触发器运行。要在分隔的 Previewer上运行另一个报表可以利用这个触发器，让你根据当前的报表值往下操作。有关的细节和其他特殊用途的报表请参看第 7章。

5.1.3 图形进程和触发器

Oracle Developer的图形进程也比表单进程简单得多，用图形作的大部分工作，特别是基本的显示对象的格式和数据库数据的联系，都是用 Layout Editor、查询和图形类型的组合来说明的。有关创建图形的细节请参看第 8章。有一些特定的操作，特别是涉及下钻 (drill-down)图形或条件判断的操作时，必须在图形触发器采用定制编程方式。图 5-16展示了图形进程，它包含几个触发器点。

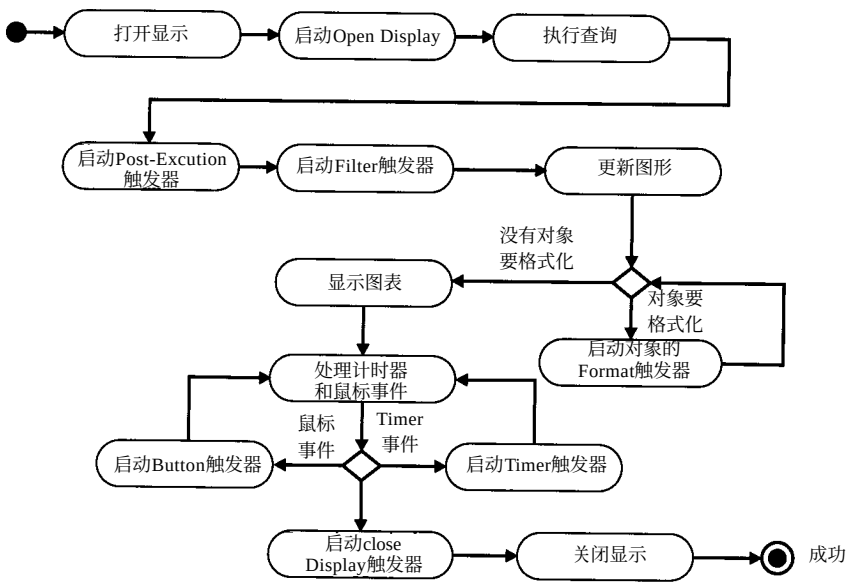

图5-16 图形进程活动图

在利用运行时图形系统打开图形显示时，为了显示而启动 Open Display触发器，然后进程执行查询，并启动Post-Execution 触发器，然后启动Filter 触发器，进程更新图形，启动那些和图形的对象有联系的 Format触发器。如果有计时器，那么在设置时间到时，计时器启动Timer触发器。如果在对象上有类似于按钮动作的鼠标事件 (例如针对下钻(drill-down)关系)，那么进程启动Button触发器。

可以建立用户自己的图形样板，作为编程格式的一种专用方法而在多个图形中使用它。可以利用各种触发器给特殊的图形显示增加专门的效果。

图形触发器

所有的图形触发器都是 PL/SQL程序(不像表单中的无记录的块)。可以通过拥有触发器的对象的属性表把任何程序连接到 Object Navigator的触发器上。有专门的库程序包 OG，它包含有一些函数，其功能是用于获得内部元素 (显示、查询、计时器等等)、格式化对象和执行不同的图形进程。

如果需要初始化图形上的某些东西，比如建立显示层、启动计时器或执行专门的查询，可以用Open Display触发器做这些事情。 Open Display触发器和Close Display触发器是Display Obiect属性表中的登记项。

如果想用某种方法处理查询数据，或者根据已有的查询建立某些东西，那么可以利用Post-Execution触发器。可以在Query属性表中建立它。

如果想根据其他的元素或查询的数据关系来格式化个别的图形元素，那么可以使用和这个图形元素联系的Format触发器。 Oracle Developer在元素的显示前执行Format触发器。可以从对象的属性表创建Format触发器。

Timer处理这样一些情况，比如想在某一时间间隔之后更新显示。一个普通的例子是，查询实时事务数据以得到全新的图形，显示出在一段时间内所发生的变化。 Timer有点复杂，应该用它们作一些试验来熟悉它们的性能。通过把 Timer触发器和查询或 Timer对象联系，可以自动执行查询和/或执行Timer触发器。

按钮过程更复杂， Oracle Developer为向下钻数据的操作自动建立相应的事件和处理。但是如果想要更复杂的事件和图形变化，那么需要把 Button触发器和图形对象联系在一起。Oracle Developer处理鼠标移动事件 (鼠标下、鼠标上、向下移动，向上移动)，可以设置对象接收这些事件。可以在设置后测试触发器程序中的事件。例如，可以在按钮图形中建立一个按钮进程，它断开计时器并自动更新显示。可以通过对象的属性表建立事件接收和创建按钮程序。

最后，还可以利用Close Display触发器清除所需要清除的任何东西。如果图形编程的其他部分发生了变化，你可以在这里把变化提交给数据库。

现在已经熟悉了Forms、Reports和Graphics的基本进程。本章的其余部分，讲述在使用它们时所要面对的一个基本的设计问题。虽然在所有的平台上，进程都是相同的，但 Oracle Developer的其他方面不一样。下一部分介绍跨平台设计的问题。

5.2 对因特网和其他平台的设计

基本工作做完以后，为了使应用程序可以在任何地方运行，还需要做些什么呢？有可能希望在不同的因特网浏览器、 Windows、X-Windows或Macintosh上使用应用程序，如何能保

证应用程序在所希望运行它的所有不同的浏览器和用户接口上都能运行得很好呢？跨平台开发有几个问题要考虑：

- 控件和菜单的标准：在目标平台上像按钮那样的控件的标准和菜单标准是什么？是否要完全遵守所有这些标准？跨平台操作的一致性比遵守平台的标准更重要吗？
- 颜色及模式：在不同的平台之间如何转换颜色和模式。
- 字体：对不同的平台如何转换字体？每个平台上的标准字型是什么？在应用程序使用的不同的监视器和打印机上的字型如何显示？
- 平台特有的功能度：可以利用任何图标、用户出口以及平台特有的特性比如 OLE或AppleEvents吗？要通过ORA_FFI使用内部的DLL吗？要使用操作系统的文件路径名特性吗？需要执行外部程序实现应用程序的某些特性吗？要使用本地平台上下文敏感的帮助吗？

注意 如果需要运行采用字符模式的应用程序，那么和以上的考虑有所不同。据Oracle公司称，在以后的版本中Oracle Developer不再支持字符模式。如果还没有安排把传统应用程序移到图形用户接口上来，那么现在是开始做的时候了。

你将发现通过计划和标准化来处理这些问题比当这些问题出现后再作处理要容易。

5.2.1 控件标准和菜单标准

每个平台都有专门的图形用户接口准则，所有的应用程序都应该遵守。 Oracle Developer自动地把标准的控制对象(按钮、复选框、单选按钮及其他)转换为适合于应用程序的设计平台的格式。

对每种平台的特殊准则请参看以下的参考资料：

- Windows 95/98 and NT4：Microsoft Corporation, The Windows Interface Guidelines for Software Design，Microsoft Press 1995.
- Macintosh：Apple Computer, Macintosh Haman Interface Guidelines，Addison-Wesley, 1992.
- Motif：Open Software Foundation, The Motif Style Guide Release 1.2 Prentice-Hall, 1992.

就我所知，没有因特网应用程序可接受的标准设计指南，因为因特网上装满了你可以想象到的任何一种设计。其他平台可以有它们自己的模式指南，你可以得到这些指南并把它作为参考资料。

现在，面临第一个大的决定是给定这些标准，你想遵守这些标准吗？通常答案是要遵守，但也不总是这样。如果你的用户要自己跨平台工作，那么应用程序的用户可能要从一个平台移到另一个平台，他们通常希望应用程序在每个平台上都一样工作。另一方面，如果用户总是使用同样的平台，但不同的用户却使用不同的平台，那么可能希望遵守平台标准，而让应用程序在不同的平台上显示出不同的特性。

下面用菜单、按钮和图标作例子。

在Windows中，MDI应用程序在MDI窗口提供单个菜单。在Motif中，每个窗口有它自己的菜单。在Macintosh中，在屏幕的上面只有一个面向所有应用程序的菜单。如果选择使用平台的标准，那么，为了与你的工作相适应，将需要一个单独的 Windows应用程序来支持MDI

方法，而对Motif和Macintosh则需要额外编码，使得能在每个窗口上正常地工作。另一方面，如果希望所有的系统都能相同地工作，那么可以在 Motif和Macintosh中进行修改，效仿MDI应用程序菜单的特性。

不同平台上的按钮差别很大。有两种基本的差别：护城河 (moats)和适航性(navigability)。在开始设计“ 吊桥 ”(drawbridge)之前，设计按钮要考虑的重点是一个“ 护城河 ”，这是一个缺省选择，它是在按回车键时执行。而适航性是利用键盘从控件移到控件的能力。 Windows的“ 护城河 ”与Motif和Macintosh的“ 护城河 ”相比是非常小的。当你从一个平台移到另一个平台时，按钮可能看起来实在奇怪。如果你的目标是坚持平台的标准，那么你必须对每个平台有不同的按钮设置。有关操作系统和图形用户接口访问信息的细节请参看后面部分的内容。

最后，图形说明了一个基本的事实：一幅图片可能胜过成千上万个字。但是不必花成千上万小时的努力。没有一个标准的图标格式。 Windows有它的ICO文件， Motif使用它自己的格式， Web页则使用GIF文件，而Macintosh把图标编译成应用程序。如果要使用图标，那么对每个平台都需要有单独的图标库，如果决定使所有的工具条和图标的按钮都相同，那么必须努力工作，使每样东西都一致。

5.2.2 颜色和模式

颜色，即使是一个点，也可能使应用程序变得活跃起来。但是使用太多或错误的颜色组合，也可能会把事情弄得一团糟。应用程序开发者近来经常会得到有关他们开发的应用程序的颜色出错的报告。当应用程序是在单色监视器上运行，或当使用者试图打印屏幕的图像时，或在屏幕上看到的关键组件出现的不是正宗的深兰色而是黑色的时候，尤其如此。如果一个平台使用标准不同的颜色，那么可能看不到类似于进度条那样的控件，因为它们可能使用了和操作系统控制的背景相同的颜色。另外，重要的是要知道许多 Motif和Macintosh用户用的是灰度级监视器，根本不容易区分颜色。所以，颜色不应该表示信息。

除非是丰富经验的颜色专家，否则限制在目标平台上使用将三、四种基本颜色混在一起的颜色和在单色或灰度设备上工作。把这些放到被命名的属性和样板上，确保所有的应用程序用正确的效果启动。测试所有目标系统上的颜色组合，确保它们工作。

同样的规则也适用于模式：不要滥用它们，因为在不同的显示中它们可能会有显著的不同。

最后，虽然Oracle Developer提供了一个256种颜色的调色板，但在个别显示中看到的实际颜色可能有所不同，有时差别还很大，特别是在 Windows中，这可能是一个实际问题。监视器的分辨率和驱动程序对颜色调色板也有影响。对于控件和窗口， Windows通过Control Panel的桌面颜色设置来控制颜色调色板。再次提醒你，限制你的颜色范围，并在目标显示中测试它们以发现问题。

5.2.3 字体

在平台之间转换字体仍然是相当困难的。 Oracle Developer做了它能做的一切，但最终它依赖于为可移植性所规定的转换方式。

所应该做的第一件事是建立 Coordinate System和栅格。理想的情况是把这些设置放在样板里，或者在创建新表单时建立它们。这个设置以后的变化可能会在所建立的布局中引起问

题，因为Oracle Developer用新的计量单位重新计算坐标。为了在平台之间有最大的可移植性，应使用厘米或英寸表示实数坐标。也可以使用点，它提供比较文本和控件的大小的能力，因为Oracle Developer总是用点来表示文本的大小。我发现使用英寸比较容易操作，因为我习惯于这种计量。然而在Windows中，它允许不同的屏幕分辨率，当在分辨率差别很大的监视器上查看应用程序时，一英寸的差别并不很大。为了设置这个选项，选择表单对象并显示它的属性表。找出Coordinate System属性，并在More按钮上单击，显示Coordinate Info对话框(见图5-17)。

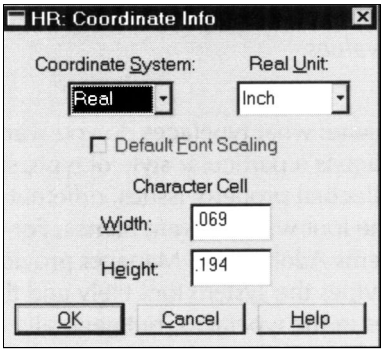

图5-17 Coordinate Info对话框

把Coordinate System属性设置为Real，把Real Unit设置为英寸或厘米，如果使用象素将使得在系统间移植应用程序很困难。现在，在每个画布中需要建立栅格。对于一个很细的栅格，把它设置为1/4英寸，在每个单元中有四个归整 (Snap)点。这就意味着可以把背景文本调整到一英寸的1/16以内，这对于可移植性来说，是一个适当的误差余量。要建立栅格的标尺，应在Layout Editor中显示画布，并选择Format |Layout Options | Ruler菜单项，选择这个项显示Ruler Settings对话框(见图5-18)。

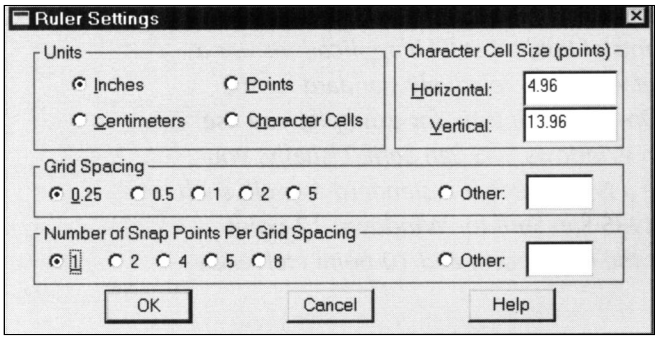

图5-18 Ruler Settings对话框

把标尺的Unit (单位)设置成与所选择的Coordinate System同样的单位(英寸或厘米)，把Grid Spacing (表格空间)设置为0.25，把Number of Snap Points Per Grid Spacing (每个表格空间的抽点数目) 设置为4(每1/4英寸4个抽点)。

提示 也可以使用Character Coordinate System。但由于字符的大粒状性质可能导致在画布中调整项和背景文本有困难。如有可能，通过项的Prompt (提示) 属性来使用提示说明，在其中规定调整的属性值。

现在选择字型调色板：想使用什么样的字样？哪种类型的大小和粗细？字样是一种特殊形式的类型。比如 Times Roman或Helvetica？因为智力属性问题，不同的系统和计算机销售商用不同的名字提供本来是相同的字型。例如 Apple Computer和Adobe Systems Adobe Type Manager提供Arial。而Windows 3.1提供系统字型Helv和True Type字型Arial。字型有较小的差别，在开发数据库应用程序中并不重要。但对于像 World Wide Web页面那样的应用程序来说，它需要表示特色和创造性字型，就变得比较重要了，是不是？好了，如果你想开发一些有创造性的应用程序，试一本像“停止偷羊”那样的书， Erik Spiekermann和E.M.Ginger著

(Mountain View, CA:Adobe出版社, 1993)但是，要认识到“创造性的”和“可移植的”可能是相互不相容的，至少就字型技术而言是这样。

注意 除了像字处理程序，或者是显示大量文本的其他系统那样的应用程序以外，大多数GUI应用程序使用对平台相对标准的简单字型。例如，在 Macintosh上使用Geneva，在Windows上使用MS San Serif。通常你还希望使用一种标准的类型尺寸，比如对Windows，用10点MS San Serif，对Macintosh用12点Gevena，在Motif用 10点Helvetica。

还应该决定在什么地方和如何使用黑体字、斜体字和下面加横线的字型。大部分时候应该尽可能少用这些字型特性，应该使不同显示文本对象的类型尺寸标准化。例如，使所有的标号为10点。如果需要在不同的平台转换字型的话，这将对你有帮助。做这项工作的最好方法是使用可视化属性Named和划分子类对象库的对象。有关使用 Oracle Developer的这些再用的特性的细节请参看第10章。

有可能发现，就像颜色在不同平台上不同一样，由于字型技术的不同，一个给定的字型在不同的平台不一致。应该在所有的目标平台测试你的标准字型。

可以在系统登录中的变量 FORM60_DEFAULT_FONT为应用程序表单的样本文本设置缺省的字型。在命名的可视化属性和属性种类中执行你的标准，确保应用程序遵守标准。

如果需要在平台之间转换字型，那么可以利用 Oracle Developer字型别名文件(在Windows中是UIFONT.ALI，在Macintosh系统是Preferences文件夹的Oracle Font Aliases 文件)。这些文件通过名字简单地把一种字型映射为另外一种字型。有关平台字型映射的细节，请参看目标平台的安装指南，实验不同的字形映射，直到对应用程序的效果满意为止。不同风格画布项要求不同的字型变换。

会发现在长久的运行中使用对象库来管理平台的差异比较容易，特别是如果要对不同的对象使用几种不同标准的字型时更是如此。有关使用对象库的细节，请参看第 10章。

5.2.4 平台特有的功能性

有几件可以在Oracle Developer上作的事情，限制在单个平台上作：

- 菜单和状态条/控制台。
- 图标。
- 用户出口。
- 平台特有的特性。

每个平台对菜单都有它自己离奇的处理方法。在 Windows平台上，MDI要求属于MDI窗口的单个菜单，该窗口包含所有其他的窗口。 MDI窗口还显示控制台或状态条。 Motif窗口(和Web窗口)是独立的，不包含在一个大的窗口内，每个窗口有它自己的菜单，可以从连接的父窗口得到。Maintosh窗口没有菜单，它们使用在屏幕上面的单个菜单，当把焦点转换到窗口时，它用窗口专用菜单代替上面的菜单。有关更详细的信息请参看第 6章“菜单”部分的内容。

图标来自图标文件，图标文件的格式与平台有关。所以对应每种平台需要建立适合那种平台格式的图标文件。 Windows有它的ICO文件，Motif使用它自己的格式， Web页使用GIF文件，而Macintosh把图标编译成应用程序。

在每个平台上用户出口被编译成平台可执行的库格式。如果应用程序依赖用户出口，那么将需要把代码从一个平台转到另一个平台。

最后，有一些可以在各个平台上利用的东西，但在其他平台上无效：

- Windows：DDE、OLE、VBX和Active X。
- Macintosh：AppleEvents。
- 所有的：通过HOST内部程序调用的任何东西，还有操作图形用户界面对象的用户出口。
- 因特网：以上所有的内容，均因为标准的 Java不允许使用这些特性，但其中 Active X将来可能会变化。其他的 Web限制：没有组合框 (Java没有和它们相当的东西)，没有GIF格式的图标，没有When-Mouse-Enter，When-Mouse-Leave和When-Mouse-Move触发器(要求的网络信息通讯太多)。还应该避免使用计时器触发器，避免大量使用图像，两者都可能引起因特网应用程序的网络性能问题。

提示 和Oracle Developer一起的联机文档资料广泛地讨论因特网应用程序的开发，在设计因特网应用程序之前一定要参考这些资料。

可以在触发器代码中运用的一个技巧是引用确定可移植性问题的应用程序属性。这让你通过使用Get_Application_Property内部函数检查这些属性的值来选择运行时使用哪些属性值：

- DISPLAY_HEIGHT和DISPLAY_WIDTH：(显示的高度和显示的宽度)这些这是告诉你当前的显示有多大，单位取决于是如何建立表单 Coordinate System的，这是表单模块的一个属性。
- OPERATING_SYSTEM(操作系统)：这个变量告诉你应用程序当前正在其上运行的平台的名字(MSWINDOWS、WIN32COMMON、MACINTOSH、SunOS、VMS、UNIX或HP-UX)。
- USER_INTERFACE(用户接口)：这个变量告诉你应用程序当前正在其上运行的用户接口技术的名字。(WEB、MOTIF、MACINTOSH、MSWINDOWS、MSWINDOWS32、PM、X、VARCHR2MODE、BLOCKMODE或UNKNOWN)。

5.2.5 标准和计划

为了可移植性必须做的一件事情是决定支持哪些平台。如果可以避免使用一些非常不同的平台，比如字符模式终端，那么会发现很容易达到可移植性。如果可以对一些应用程序进行一些限制，也可能有帮助。例如，如果销售汽车的领导因跟踪销售管理应用程序需要使用膝上型计算机，而其他人都使用 17英寸的SVGA终端，这样，只限制两个膝上机应用程序的窗口的尺寸就可以节省许多工作。在任何情况下，设计表单和图形的布局都需要知道窗口尺寸的变化。在某些情况下，也可以利用适当的滚动方式来适应较小的屏幕。请参看下一章的有关滚动画布和层叠画布的能力的内容。

一旦明确了所使用的平台，就把涉及到可移植性的用户接口对象和属性的标准放在一起。如果充分利用指定的可视化属性，属性类和图形样板能够使所设定的标准很容易在工作的应用程序中实现。那样一来，会发现把应用移到不同的平台是非常容易的。有关通过对象库、划分子类、指定可视化属性和属性分类(和其他方法)增加代码的可再用的细节，请参看第10章。

概括地说，关于可移植性问题，利用 Oracle Developer创立可移植的应用程序相对来说比较容易，但是要制定计划和标准才能有效和高效地工作。

现在已做好了准备，通过利用 Oracle Developer的高级编程和开发特性把实验模型带到下一阶段。接下来的部分研究更高级的问题，它们将显著增加创造安全稳固的应用程序的效率。