第9章　迁移到体系结构设计环境

在当今现实环境中，若想仓促地实现任何一种体系结构，注定都是要失败的。这其中存在许多风险，企业可能要等很长的时间才有可能得到回报，以及由于试图想以后不再做任何变化，不想采用任何渐进式的途径，而是采用革命式途径所带来的不现实性。

有一点确实很不错的是，迁移到体系结构设计的数据仓库环境中的过程，是一个逐步的，每次只需完成有限的可提交的迁移工作。实现得最为成功的体系结构设计环境，是那些以每次一遍的方式建立的数据仓库环境。这样，建立数据仓库只需要最少的人力资源，对现存应用环境造成的破坏也必定是最小的。对这种重复式的开发而言，开发规模和速度都很重要，结果也必须能够快速交付。

本章中，将讨论一种普通的迁移方案和开发方法。这种方法在附录中也有详尽的阐述。该迁移方案已经得到许多企业的成功应用，这决不是幻想飞翔。这个方法本身来源于许多企业的实践经验。当然，每个企业各自的方法会有不同的变化和置换。但这种迁移方案和方法在许多企业都已取得了很大的成功，这点非常有益于树立各方面开发者的信心。

9.1　一种迁移方案

这种迁移方案的起点是一个数据模型。数据模型描述企业的信息需求。它指出一个企业所需要的，而并不一定是企业当前所具有的东西。在建立这个模型时，并不考虑任何技术问题。

数据模型可以在内部建立起来，或者可以在一个普通的数据模型的基础上建立起来。数据模型(至少！)需要给出如下的内容：

- 企业的主要主题。
- 各个主要主题之间的各种关系。
- 更全面地描述主要主题的键码组和属性组，包括：
  - 这些主要主题的属性。
  - 这些主要主题的键码。
  - 键码和属性的重复组。
  - 各个主要主题领域之间的连接器。
  - 图表化关系。

在理论上，建立体系结构设计环境也可以不要数据模型。然而，实际上从来没人这样做。若想不用数据模型就去建立体系结构设计环境，就像是没有地图的航行一样，或许能成功，但非常容易出现麻烦和错误。这也就像一个从未离开过得克萨斯的人，手上没有地图，也没人指路，却想要降落在纽约的拉瓜尔地亚机场，然后要开车到曼哈顿中区去一样。或许真能到那里，但直觉告诉我们，有一张地图肯定会好得多。

图9-1表明，建立或者获得一个数据模型是迁移过程的起点。

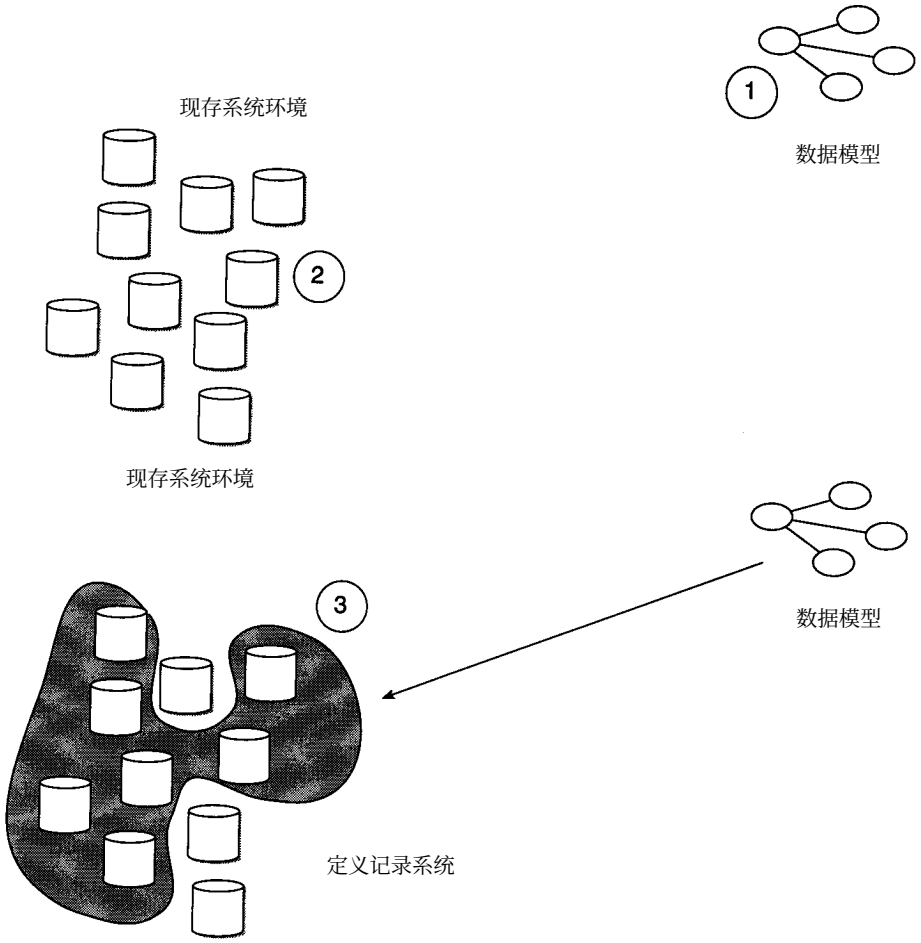

图9-1 迁移到体系结构设计环境

有了数据模型以后，下一步工作就是定义记录系统，记录系统是由企业已有的现存系统来定义的。通常，这些系统是很混乱的。

记录系统只不过是找出现存系统所具有的“最好的”数据。此时，数据模型作为决定什么是最好数据的判定基准。换句话说，数据体系结构设计人员从数据模型开始，找到手中最符合数据模型需求的数据。有时找到的结果也不是很完美。在有些情况下，现存的系统环境中找不到数据来例证数据模型中的数据。而另外一些情况下，现存的系统环境中，有许多数据源在不同的情形下为记录系统提供数据。

现存数据系统的哪些数据源“最好”是由如下的标准来决定的：

■ 现存系统环境中的数据哪些是最完备的？

- 现存系统环境中的哪些数据是最实时的？
- 现存系统环境中的哪些数据是最准确的？
- 现存系统环境中的哪些数据是与输入现存系统环境的数据源最近的？
- 现存系统环境中的哪些数据最接近数据模型的数据结构？是按键码判断，还是按各个属性或多个数据属性的组合判断？

使用上面给出的数据模型和衡量标准，分析人员就可定义出记录系统。

定义好记录系统以后，下一个步骤就是设计数据仓库，如图 9-2所示。

如果数据模型建立工作进行得很好的话，设计数据仓库的工作也必将很简单。只需要修

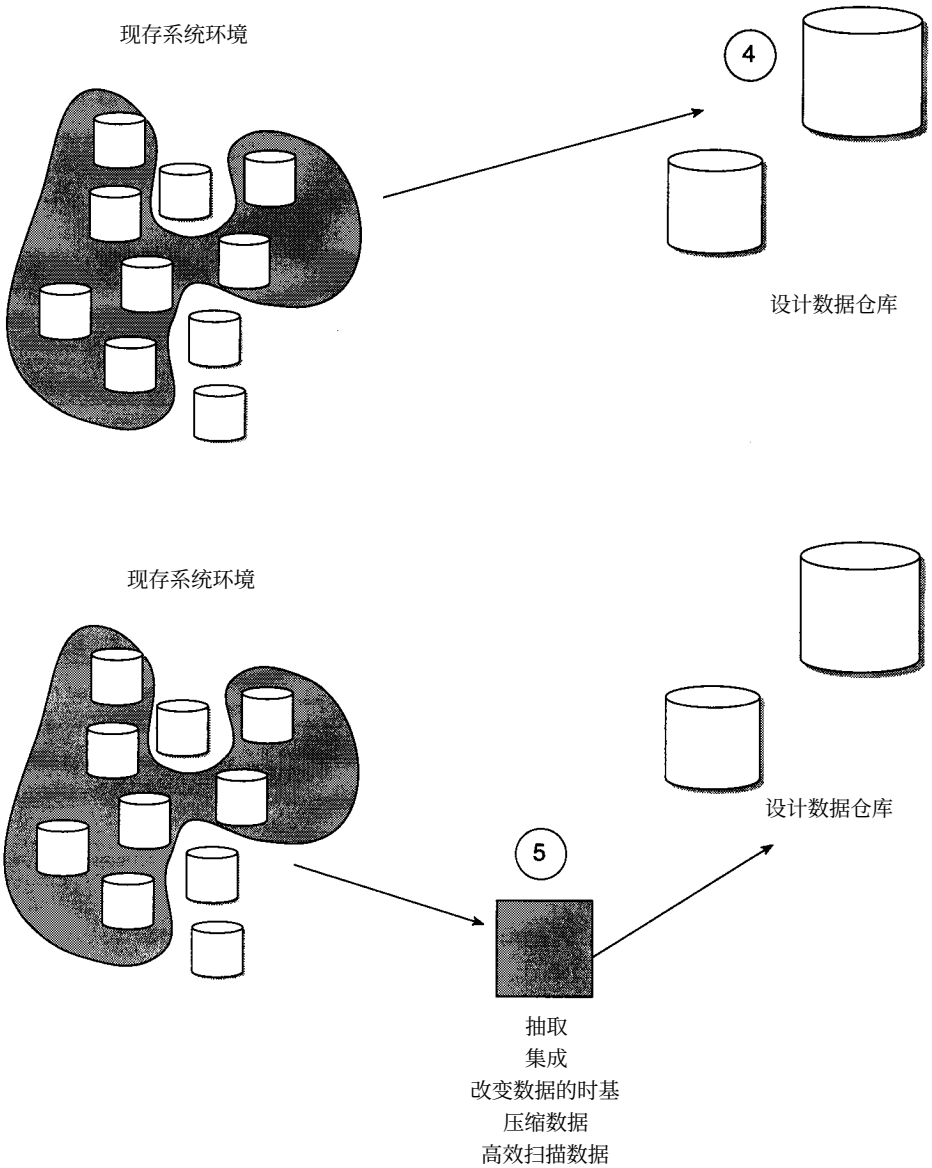

图9-2 迁移到体系结构设计环境

改数据模型的少数几个方面，就可以将数据模型变为一个数据仓库的设计。要做的工作主要有以下这些：

- ■ 如果原先没有时间元素的话，时间元素必须加入到键码结构中。
- ■ 必须清除所有的纯操作型数据。
- ■ 需要将参照完整性关系转换成“ 人工关系 ”。
- ■ 将经常需要用到的导出数据加入到设计中。
- ■ 为了适合于以下各项要求，需要对数据的结构进行调整：
  - • 增加数据阵列。
  - • 增加数据冗余。
  - • 在合适的情况下进一步分离数据。
  - • 在合适的时候合并数据表。
- ■ 需要做数据的稳定性分析。

一旦要设计数据仓库，就必须按主题领域进行组织，典型的主题领域有：

- ■ 客户
- ■ 产品
- ■ 销售
- ■ 账目
- ■ 活动
- ■ 装运

在主题领域内，有许多独立的数据表，每个均以一个公用键码连接。

数据仓库设计好以后，下一步就是设计和建立操作型环境中的记录系统和数据仓库之间的接口，这些接口能保证数据仓库的载入工作能有序地进行。

乍看起来，这些接口似乎仅仅是一个数据抽取过程。当然，数据抽取过程确实是在此进行的，但是，在接口中还包括了许多其他工作：

- ■ 来自操作型、面向应用型环境的数据的集成。
- ■ 数据时基的变更。
- ■ 数据压缩。
- ■ 对现存系统环境的有效扫描。

其中的多数问题已经在本书的其他部分讨论过了。

有意思的是，建立一个数据仓库所需要的大多数开发资源都花费在这点上了。建立数据仓库所需的80%的精力都花在此处并非不正常。在安排数据仓库的开发工作时，许多开发者过高地估计了其他工作所需要的时间，而过低估计了设计和建立操作型环境与数据仓库环境之间的接口所需的时间。

一旦设计并建立好了接口程序，下一步工作就是开始载入第一个主题领域，如图 9-3所示。

此时，有很多原因使得只需要载入一部分数据仓库所需要的数据。很可能需要对数据做必要的改变。只载入一小部分的数据也就意味着可以简单快速地完成那些需要进行的改变。大规模地载入大量的数据会丧失数据仓库的灵活性。一旦最终用户有机会观察数据，并向数据体系结构设计人员反馈情况，载入大量数据就可以安全进行了。

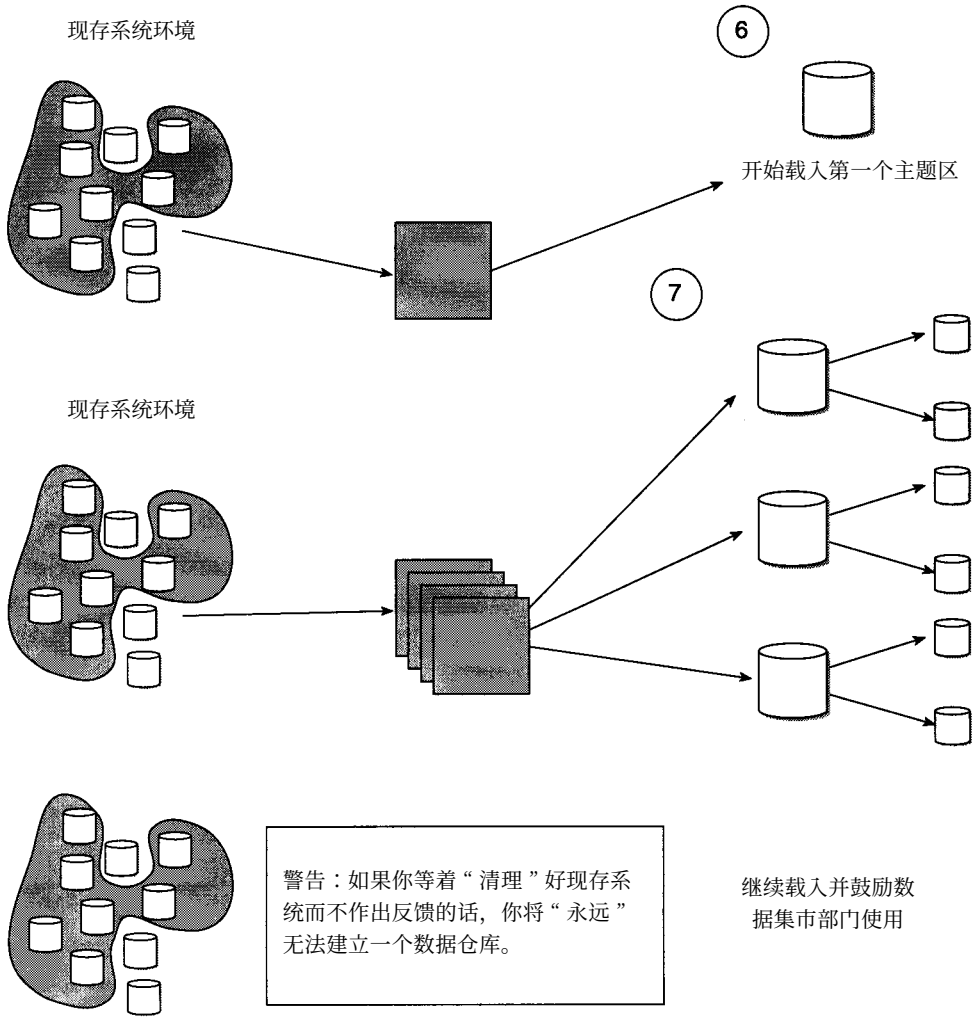

图9-3 迁移到体系结构化环境

载入和反馈过程会持续一段很长的时间(没有一定的限期)。另外，数据仓库中的数据在此过程中也在不断地改变。

这时，必须提出一点警告：“如果你等着清理好的现存系统，你将永远无法建立一个数据仓库。”现存系统的操作型环境下的问题和活动必须独立于数据仓库的环境下的问题和活动。

在这点上，一件值得做的事情是观察数据仓库中数据的刷新频率。通常，数据仓库中数据的刷新频率不应低于每24小时一次。在装载数据的时候，确保数据起码有24小时的时延，数据仓库的开发者就能将数据仓库变为操作型环境的可能性减小到最小程度。数据仓库服务于企业的DSS的需要，而不是日常业务运作的需求。许多操作型处理依赖于数据存取瞬间的准确性(数据的当前值)。只有通过确保(至少)有24小时的时延，数据仓库开发者才有最大的成功机会。

9.2 反馈循环

数据仓库开发成功的关键是数据体系结构设计者和DSS分析者之间的反馈循环,如图9-4所示。

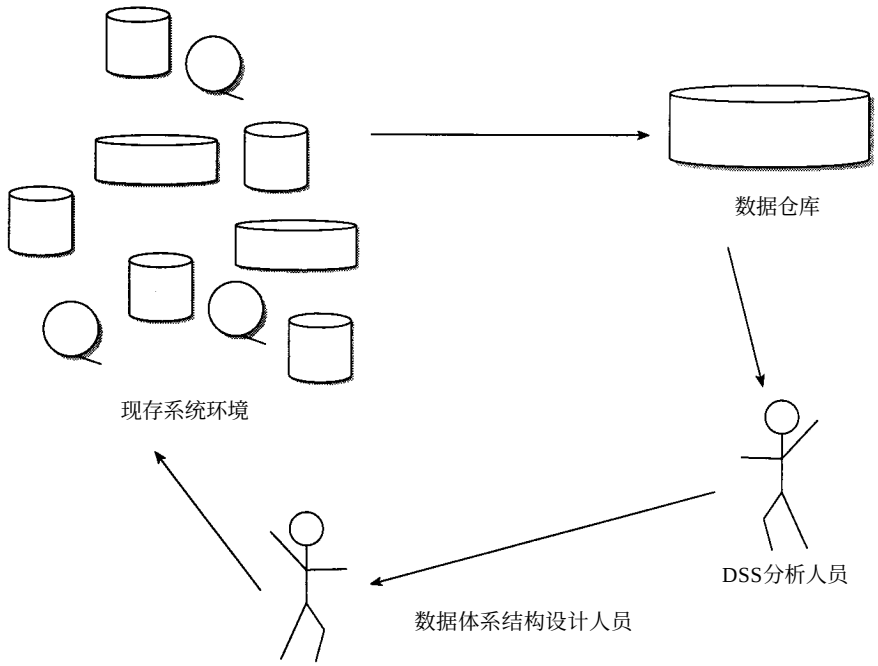

图9-4 DSS分析人员与数据体系结构设计人员之间的关键的反馈循环

图9-4表明，数据仓库是从现存系统载入而来的。 DSS分析人员将数据仓库作为分析的基础。在发现新机会的过程中， DSS分析者将那些需求反过来传送给数据体系结构设计人员，以便使他们再去做出适当的调整。

有关这种反馈循环，有几点观察结果对数据仓库环境的成功建立是至关重要的问题：

- DSS分析人员一定要严格遵循“给我我所想要的东西，然后我能告诉你我真正需要的东西”的工作模式。
- 反馈循环的周期越短，越有可能成功。
- 需要调整的数据量越大，反馈循环所需要的周期就越长。

9.3 策略方面的考虑

图9-5表明，图中所描述的各个活动的路径强调了企业的 DSS需求。

设计和建立数据仓库环境的目的是用于为企业的 DSS需求提供支持，但除DSS外，企业也有其他方面的需求。图9-6表明，企业也有操作型需求。

如图9-6所示，其中的操作型环境处于一种混乱状态。操作型环境中有许多未集成的数据，其中包含的数据和系统都已很老了，有很多补丁，已经无法再对它们进行维护了。原来确定的操作型应用的需求，已经改变得让人几乎无法识别了，等等。

前面所讨论的迁移方案仅仅适用于 DSS部分，但是，在创建数据仓库的同时，有没有可能去矫正一些或许多的操作型环境中的“混乱”呢？答案是在某种程度上有可能做一些比像美化操作型环境更少的一些重建工作。

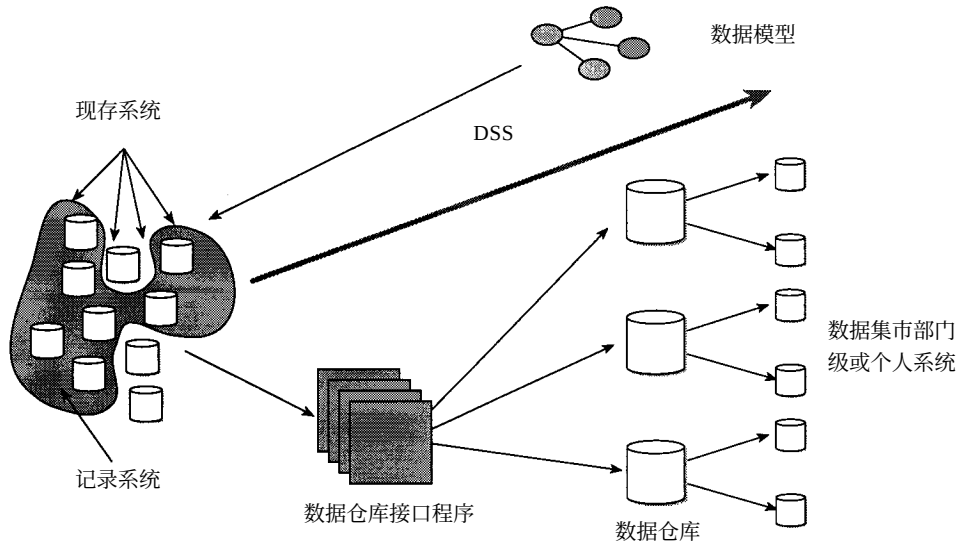

图9-5 需遵循的首要路径是DSS路径

有一个方法，它是数据仓库环境的迁移中的一个独立途径，就是以数据模型为指导，告诉管理者需对操作型环境需做的重大调整。但业界以往的记录表明，这种方法并不让人感到乐观。它所需的工作量、资源的数量以及在进行大量的重写和重建操作型数据和系统时对终端用户造成的破坏，都使得管理层很少愿意花费如此多的投入和资源去支持这种努力。

一个更好的方法是将重建操作型系统的工作和被称为“变化动因”的因素协调起来进行

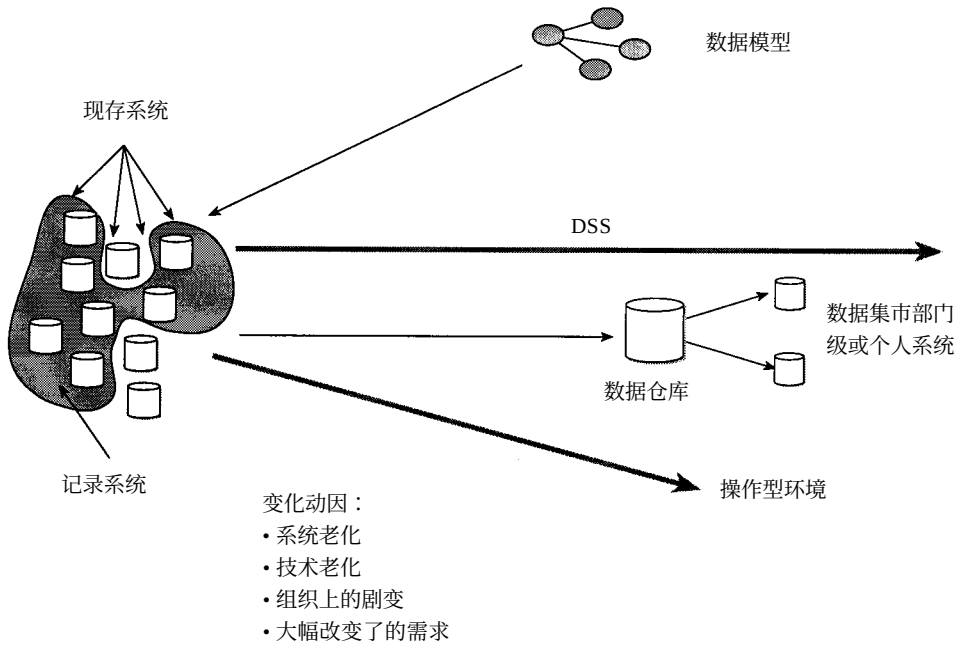

图9-6 要取得成功的话，数据体系结构设计人员应该等待，直到各个变化动因变得迫切以后，将与体系结构设计环境有关工作与合适的动因结合起来

考虑，这些变化动因有：

- 系统的老化。
- 技术的剧烈更新。
- 组织上的剧变。
- 巨大的商业变化。

面对这些由变化动因所造成的影响，管理层毫无疑问需要做出相应的变化。唯一的问题是要多快和花费多少钱。数据体系结构设计人员得将变化动因与体系结构的概念结合起来，并以此给管理层提供充分的理由，以实现操作型处理环境的重建。

数据体系结构设计人员采取的重建操作型环境的步骤如图 9-7所示，这是建立数据仓库的一项独立的活动。

首先，创建一个差异列表，这个差异列表给出了操作型环境和数据模型所描述的环境之间的差别评估。差异列表是一个简单的列表，没有很详细的描述。

下一步是影响分析。此时，对差异列表中的每一项会造成的影响都做出一个评估。有些项造成的冲击可能很严重，而其他的一些项对企业的运作所造成的影响几乎可以忽略。

再下一步，需要做出资源估计，以确定满足修复该差异列表项所需的资源数量。

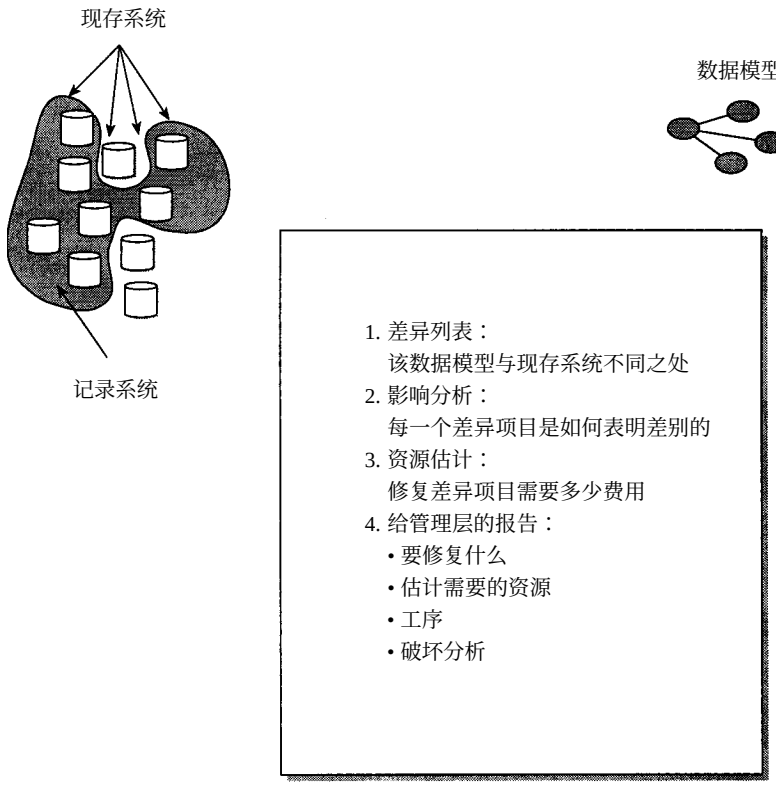

图9-7 创建操作型环境清理方案的第一步

最后，将所有以上的这些做成一个报告，提交给信息系统管理层。由管理层决定哪些工作需要进行，开展步伐如何，等等。

9.4 方法和迁移

本书的附录中讨论了一个构造数据仓库的方法。实际上，该方法的范围相当大，因为其中不仅包含如何建立数据仓库，而且也说明如何使用数据仓库。另外，其中也包含了开发操作型环境的典型工作方法，这些方法形成了所谓“数据驱动”的方法学。

讨论的方法在几个方面与迁移路径有所不同。迁移路径动态地描述总体工作步骤。而此方法讨论特定的工作步骤，从这些工作得到的可提交的数据以及这些工作的次序。但并没有叙述数据仓库创建的循环往复的动态过程。换句话说，迁移方案从三个角度描述一个概要的方案，而些方法则从一个角度描述一个详细的方案。两者结合在一起形成了一个完整的创建数据仓库所需的工作的描述。

9.5 一种数据驱动的开发方法

对开发方法需求是很普遍的。要求研究人员给出一个方法，给开发者指引一条合适的道路，指出需要做些什么、按照什么次序做、整个工作需要多长时间。虽然方法学概念本身很有吸引力的，但业界的记录并不让人满意。对方法(数据仓库或其他)的热情与对应用的失望交织在一起。对方法学的历史记录表明，从方法学到应用实现是一条很不平坦的路。

为什么这些方法让人失望呢？其中有很多的原因：

- 这些方法通常给出一个单调的、线性的工作流。实际上，几乎任何方法都需要循环重复执行。换句话说，执行二三步以后、停止、再全部或部分重复前面的步骤，都是很正常的。通常，这些方法本身没表明或没有考虑到它们本身需要重复进行一个或多个步骤。对于数据仓库而言，这种不支持重复工作的缺点会使得这种方法成为一个大问题。
- 通常，这些方法给出了一些出现或仅出现一次的工作。确实，有些工作只需做一次(当然得成功)就行了。然而，其他的一些工作在不同的情况下需要重复地做多遍(在这里指的情况不同于求精的重复过程那种情况)。
- 通常，这些方法规定好了一组需要做的工作。在许多情况下，其中有些根本就用不着做，而有些需要做的工作却没有在方法中列出来，如此等等。
- 这些方法经常说该如何做些工作，而不是说需要做什么。在描述如何做些什么的时候，这些方法在碰到细节和特殊情况时，其有效性就成问题了。
- 这些方法对要开发的系统的规模不加区分。有些系统很小，严格的方法在此时就没什么意义；有些系统或许正好与某个方法相适应；而有些系统非常大，它们的规模和复杂性会使方法根本就不起作用。
- 这些方法经常将项目管理问题与需要做的设计和开发工作混在一起。通常情况下，项目管理问题应该与开发方法相关问题分开。
- 这些方法经常对操作型处理和DSS处理不加区分。操作型处理和DSS处理的系统开发生命周期在许多方面是正好相反的。要取得成功，一个方法必须区分操作型和 DSS的处理和开发。
- 出现失败的情况下，这些方法一般都没有检查点和停止处。“如果前面一个步骤没有正确执行的话，下一步该怎么办呢？”，方法中通常不把处理此类问题作为一部分。

- 这些方法常常是作为解决方案，而不是作为工具出售。当这些方法被当作解决方案来出售时，不可避免地，一些其他的很好的判断和常识就可能会被这种方法所替代，而这经常是错误的。
- 这些方法总是提交出非常多的论文，却鲜有设计工作。设计和开发工作的地位被论文不合理地取代了。
- 这些方法可能很复杂，预计到了或许曾经发生过的或可能会发生的各种可能性。

尽管有这些缺点，对通用方法的需求仍然存在。附录中将会讨论一个适用于数据驱动环境的通用方法，该方法充分考虑到了这些方法的缺陷和以往的记录。以概要方式给出的这种数据驱动方法，很大程度上要归功于一些研究这种方法的先驱者。为此，想要得到对方法中所讨论的复杂性和技术更充分的阐述的话，请参考书后所列出的参考文献。

数据驱动方法有一个突出的方面，就是它是建立在前面进行的工作的基础之上，也就是建立在原先已开发的代码和处理的基础之上。基于原有工作之上的开发要获得成功，唯一的途径就是要找出共同性。在开发者输入第一行代码或设计第一个数据库之前，他们应该知道什么东西现在已经有了，它们对开发过程的影响如何。在使用已经存在的东西的时候，必须保持清醒的头脑，而不要做重复工作。这就是基于数据驱动的开发的实质。

9.6 数据驱动的方法

为什么一种方法叫做数据驱动的方法呢？数据驱动的方法与任何其他方法有什么区别？数据驱动的方法起码有如下两个显著的特点：

- 数据驱动的方法不是按照一个应用接一个应用的方法去开发系统的。相反地，原先已建立好的代码和数据被作为新代码和数据的基础，而不是新老并立。要建立在原先的成果之上，就必须找出数据和处理的共同性。一旦找出共同性，已有的数据就可以作为基础，若不存在任何数据，则需建立新的数据，而这些新建立的数据或许又可以作为以后应用的基础。找出共同性的关键就是数据模型。
- 必须强调的是，数据应集中存放，形成数据仓库，作为 DSS处理的基础，要认识到DSS处理与操作型环境相比，有一个大不相同的开发生命周期。

9.7 系统开发生命周期

操作型和DSS系统的开发生命周期之间的深刻区别，从根本上体现了数据驱动开发方法的特点。操作型系统的开发生命周期特点是，它开始于需求，结束于代码；而 DSS处理的开发生命周期的特点，则是开始于数据，而结束于需求。

9.8 一个哲学上的考虑

在某些方面，有关方法的最好的一个例子是男女童子军的荣誉徽章体制。该荣誉徽章体制用来衡量队员们什么时候应该晋升一个等级。该体制既应用于住在乡村也用于住在城市的男孩和女孩，不管是喜欢体育的还是好学习的，也不管地域如何。简言之，荣誉徽章体制是一种统一的、用于衡量成就的、经受了时间考验的方法。

荣誉徽章方法有什么秘密可言吗？如果有，那就是：荣誉徽章方法并不给你规定任何一样工作该如何完成，相反，它只是仅仅说明该做些什么东西，以及给出一些衡量成就的不同

参数。“该怎么做”这个问题则留给了男女童子军们自己。

在哲学上，本书附录所给出的方法采用了与荣誉徽章体制相同的观点。一般地说，其中描述了需要达到的各个目标，以及各个工作的次序。该如何得到所需结果，则完全留给了开发者。

9.9 操作型开发/DSS开发

数据驱动方法将分三个部分给出：方法 1、方法2和方法3。方法的第一部分即方法 1是用于操作型系统和处理的，习惯于传统的结构化的操作型方法的人对这部分的方法或许很熟悉。方法2是用于DSS系统和处理的，也就是数据仓库的，这部分方法的本质在于将数据模型作为媒介物，以它为中心找出数据的共同性。正是在附录的这部分描述了数据仓库的开发方法。方法的第三部分即方法 3描述开发过程中的启发式部件中会出现什么，其中还说明数据仓库的用法。

这三部分构成了数据驱动开发方法。

9.10 小结

本章探讨了一种迁移方案和一种方法(见附录)。该迁移方案讨论了将数据从现存系统环境中转移到数据仓库环境的相关问题。另外，也讨论了操作型环境该如何组织的问题。

本章还讨论了一个通用的数据驱动的方法，该通用方法有三个阶段：操作型阶段、数据仓库构造阶段和数据仓库重复使用阶段。