软 件 工 程 系 列

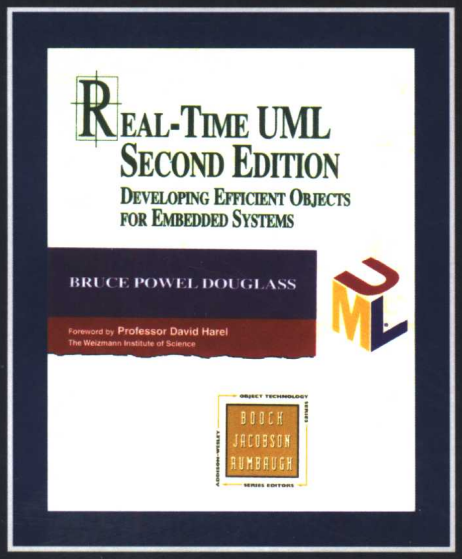
Addison Wesley

Real-Time UML Second Edition
Developing Efficient Objects for Embedded Systems

实时UML——
开发嵌入式系统高效对象
（第二版）

状态图发明人、以色列魏茨曼科学学院教授
David Harel
作序并大力推荐

[美] Bruce Powel Douglass 著
尹浩琼 欧阳宇 译



中国电力出版社
www.infopower.com.cn

软 件 工 程 系 列

Real-Time UML Second Edition
Developing Efficient Objects for Embedded Systems

实时UML
开发嵌入式系统高效对象
（第二版）

[美]Bruce Powel Douglass 著
尹浩琼 欧阳宇 译

中国电力出版社

Real-Time UML:Developing Efficient Objects for Embedded Systems,2nd Edition (ISBN 0-201-65784-8)
Bruce Powel Douglass

图书在版编目（CIP）数据

实时 UML——开发嵌入式系统高效对象（第二版）/（美）道格拉斯；尹浩琼，欧阳宇译.
—北京：中国电力出版社，2003
（软件工程丛书）
ISBN 7-5083-1812-9
Ⅰ.实... Ⅱ.①道...②尹...③欧... Ⅲ.面向对象语言，UML—程序设计 Ⅳ.TP312
中国版本图书馆 CIP 数据核字（2003）第 090546 号

丛 书 名：软件工程丛书
书　　名：实时UML——开发嵌入式系统高效对象（第二版）
编　　著：（美）Douglass
翻　　译：尹浩琼　欧阳宇
责任编辑：陈维宁
出版发行：中国电力出版社
　　地址：北京市三里河路6号　　邮政编码：100044
　　电话：（010）88515918　　传　　真：（010）88518169
印　　刷：汇鑫印务有限公司
开　　本：787×1092　1/16　　印　　张：16.5　　字　　数：360千字
书　　号：ISBN 7-5083-1812-9
版　　次：2003 年 12 月北京第 1 版　　2003 年 12 月第 1 次印刷
定　　价：35.00 元

前　言

嵌入式计算机系统，以及反应式和实时系统的时代已经到来。读完本书，你就会发现嵌入式系统无处不在；除了传统的桌面和膝上计算机，还有更多的计算机隐藏在无数的机器或设备里面。

哪里有计算机和计算机处理系统，哪里就有驱动它们的软件。软件不是从树上长出来的，而是人们不断地编写、理解、分析、使用、维护和更新的结果。是人的编程唤起了在抽象层次上建模复杂系统的需要，这比“一般”的编程语言要高一个档次。这也产生了对能引导软件工程师和编程人员处理建模过程的方法学的需要。

大家一致认为好的图解是设计高级建模方法所需奋斗的目标之一。虽然其他的东西同样重要，但是图片通常比文本或符号更容易理解。然而我们也并不是仅对图片或图表感兴趣，因为构建复杂的软件不只是人的活动。我们对图解语言感兴趣，而这些语言需要对验证和分析的计算机化支持。正如高级编程语言不但需要编辑器和版本控制工具，而且还（并且是支配性地）需要编译器和调试工具那样，建模语言不但需要漂亮的图形、文档生成工具和项目管理辅助，还需要执行模型，合成代码以及真假校验所需的手段。这意味着我们需要视觉形式主义（visual formalism），这种形式主义是由语法和语义完善起来的，其中语法决定什么被允许，语义决定被允许的东西意味着什么。这种形式主义应该尽可能地可见（很明显，有些东西不会自动地让人看到），其中重心放在图解实体之间的拓扑关系上，然而作为次佳选项，也可以是几何和度量，也可能是图标。

在历史的长河中，曾经有两种主要的高级建模方法：结构化分析（structured analysis，SA）和面向对象（object orientation，OO）。这两种方法在提出概念到开始发展有十年的差距。SA，在20世纪70年代末由DeDarco、Yourdon等人提出，将经典的过程编程概念提升到建模层次，并且完成图形化。于是就产生了通过功能分解和信息流实现，由（层次结构）数据流图描述的建模系统结构。至于系统行为，在上世纪80年代早期和中期曾出现过几个方法学团体（比如Ward/Mellor、Hatley/Pirbhai和I-Logix的STATEMATE小组），他们提出了很多详细的建议，利用基于状态图或比状态图更丰富的语言捕获行为的手段丰富了基本的SA模型。我们应该添加的精心定义过的行为建模，对于嵌入式、反应式和实时系统尤其关键。

OO建模（通常叫做OO分析和设计，或者OOAD）起源于20世纪80年代末，并且

它的历史在某种程度上与 SA 有些类似。系统结构的基本思想是将概念从面向对象编程提升到建模层次上，并且也是图形化完成的。因此，Booch 方法、OMT 和 ROOM 方法，以及很多其他方法中的对象结构模型开始处理类和实例、关系和角色、操作和事件、聚合和继承。可见性是通过将该模型建立在修饰过和浓缩过的实体－关系图的基础上获得的。至于系统行为，大多数 OO 建模方法采用了状态图语言（一种决策，即签过名的人不能再声称自己对此不清楚）。状态图与每个类都有关联，它的任务是描述出实例对象的行为。结构和行为之间微妙和复杂的连接（即，对象模型和状态图之间的连接）被 OO 方法学家们认真对待，从极度的不重视到足够重视。当然，测试要搞清楚结构和行为的语言以及它们的连接是否被充分地定义，以便支持高级模型的“解释”和“编译”——也即是说，完整模型的执行和代码合成，最终（并且很有希望）还有对需求的完全正式的校验。这只有在两种情况下才能实现，即在 ObjectTime 工具（基于 Selic、Gullekson 和 Ward 的 ROOM 方法）以及 Rhapsody 工具（来自 i-Logix，基于 Gery 以及可执行对象建模上签名人的工作）中。

在系统建模方面，SA 和 OO 范例之间的共同点越来越少。在过去的四到五年里，很多 OO 方法学家携起手来共同努力。他们交换意见，讨论问题，将各种 OO 建模方法的优点结合在一起，最终合作制订出了一个通用的统一建模语言，或者简称 UML。这场怀念 Algol60 和 Ada 的运动是在对象管理组的资助和 Grady Booch（Booch 方法的创始人）、Jim Rumbaugh（OMT 方法的开发者之一）及 Ivar Jacobson（用例之父）的领导下进行的。UML 0.8 在 1996 年发布，它有很多东西需要补充，并且很模糊，与人们期望的良构相差很远。一年之后，在很多公司的方法学家和语言设计者的帮助下，UML 小组终于推出了更严谨和更坚固的 1.0 版本。此后还出现了很多版本的 UML。1997 年，UML 被对象管理组（Object Management Group, OMG）采用为标准。如果再下些功夫，除了只是官方确认的标准之外，它还很有希望成为面向对象编程中的建模机制。这绝不是小事，因为越来越多的软件工程师意识到软件最好是用 OO 风格开发。

对于系统结构的捕获，UML 确实采用了基于实体-关系方法的类和对象的图解语言。对于早期行为分析，它推荐了用例，并使用了顺序图（通常称为消息顺序图，或者 MSC）。对于行为的完整可构造规范，它采用了状态图，这在前面的可执行对象建模工作中已经修改过了。

Bruce Douglass 做了一项很有意义的事情，他将自己的工程经验提供给那些必须构建复杂软件（尤其是实时、嵌入式和反应式软件）的人。况且，他是通过将 UML 用作底层的载体来完成这一点的。如果给定最近的 UML 标准化以及它们的快速扩展用法，那些总是担心这种系统不能快速和平稳开发的人会从中受益非浅。

另外，本书思路清晰，文笔流畅，读者看了之后会信心大增。之所以有这种感觉，主要是因为作者不像专业的方法学家那样高高在上使人敬而远之，而是紧密结合自己在

工程中的实践经验，娓娓道来，使人倍感亲切。这种区别可称为“系统行为的宏大二元性”。我们还没有能理解这种二元性的好算法。状态图看上去很适合对象内的规范，而顺序图则很好地指定了用例，但是它们的功能太弱，不能充当完全对象间规范这一角色（例如，它们不能指定“反场景”——可以禁止的场景）。最近有人提议扩展顺序图，以便它们可以捕获更多的场景，但是评审团还没有参与进来。同样，我们也没有好的算法来理解建模的两个模式之间的二元性。我们不知道如何从一个视图派生出另一个视图，不知道如何有效地测试这两个视图中提供的描述是否互相一致。

随着 UML 的流行，关于这方面的各种书籍、文章、报告、研讨会和工具一定会泛滥起来。所以读者一定要仔细甄别，从中找出真正有价值的东西。我深信 Bruce 的书就属于这一类。

至于 UML 本身，我们必须记住一点：UML 目前还有点过于庞大。我们只了解了其中的一部分；其他部分的定义还没有被详细制订出来，因此我们还不太清楚它们和 UML 可构造性核心（类图和状态图）的关系。例如，用例以及与其关联的顺序图和协作图对于试图按照场景找出系统所需行为的用户和需求工程师毫无价值。在用例领域中，我们为所有相关对象只描述了一个场景（或者紧密相关的场景的一个簇）——我们可以称之为对象间行为。与此形成鲜明对比的是，状态图描述了一个对象的所有行为——我们称之为对象内行为。

其他严重的挑战仍然存在，并且我们只解决了皮毛而已。这样的例子包括利用 UML 提供的高级手段建模的面向对象的软件的真正的、正式的校验，自动的、赏心悦目的、结构增强的 UML 图布局，处理包含离散、连续部件在内的混合系统的令人满意的方法等等。

作为处理复杂软件的一个通用方法，面向对象的方法必不可少。也许 UML 也是如此，不过我的个人感觉是随着成为建模软件标准的新鲜感的慢慢消退，UML 将变得越来越小和严密。或者，变得太臃肿，而不能真正使用。我认为它会逐渐收缩，只留下三到四种真正有用的图，其余的逐渐被人遗忘，并最终完全消失。

OO 是分析系统以及编程的强大且明智的方法，并将在很长一段时间内成为那些有自尊心的软件工程师必需的知识结构的一部分。本书将帮助他们做到这一点。另一方面，OO 并不能解决所有的问题，UML 也不能，仍然有很多工作需要完成。事实上，我们可以毫不夸张地说，我们未知的和做不到的远比我们已知的和能够做到的多。即使这样，我们现在所知道的仍然比我们几年前所希望知道的要多得多，所以我们应该时刻抱着感激和谦虚的心态。

以色列魏茨曼科学学院
数学和计算机科学系主任
David Harel 教授于 Rehovot
1999 年 7 月

译 者 序

三位面向对象方法学的创始人 Rumbaugh、Booch 和 Jacobson 共同合作，创立了 UML（统一建模语言）。自从 1997 年 UML 被 OMG 采用为标准以来，历经多次修订。从 UML 1.0 一直发展到 1.4，但就在译者翻译本书时，OMG 已经通过了 UML 2.0 提案。

统一建模语言将为软件开发商及其用户带来诸多便利。美国等计算机技术发达国家已有大量的软件开发组织开始用 UML 进行系统建模，学习和使用 UML 已经成为一种潮流。我国软件界对 UML 也相当关注，许多研究人员和技术人员已在几年前就开始了对 UML 的学习和研究。但由于 UML 的复杂性，仅通过 UML 的标准文献和国内目前的关于 UML 的资料来掌握使用它不是一件轻松的事。为此，我们专门翻译了 UML 的代表作之一，也就是本书，以帮助那些急切地想了解 UML 的读者。

本书按照大多数开发项目都遵循的分析→设计→实现等步骤来组织。第 1 章介绍了什么是实时系统和对象，以及 UML 在实时系统中的应用。讨论了 UML 中的面向对象技术，以及类和对象间的关联。类之间存在着五种关系：关联、聚合、组成、泛化和依赖关系。本章同时还简单地提到了 UML 的表示法。为后面章节的学习打下了基础。

第 2、3、4 章可算作一个部分，即分析。其中第 2 章讲述了实时系统的需求分析，第 3 章讲述了实时系统的对象结构分析，第 4 章讲述了实时系统的对象行为分析。

5、6、7 章又是一部分，主要讲的是设计。根据所作决策的作用域，可以将设计分为三个部分：体系结构设计、机械设计以及详细设计。5、6、7 章分别讲述了这三个阶段的设计。

到这里为止，实时系统中 UML 的知识已经全部介绍完了。但为了方便开发者在开发大型复杂系统时参考，附录 A 还提供了一个详细的表示法列表，虽然这些表示法在正文中都出现过。附录 B 讲了 UML 的前景。由于本书英文版在 1999 年就出版了，所以其中作的很多预测与展望，都不幸成为了历史。有兴趣的读者，参考相关资料，看看作者预测的是否准确。

本书由尹浩琼、欧阳宇主译，参与翻译工作的人员还有石朝江、李明、周刚、谢俊、谢小花。由于译者水平有限，书中错误与疏漏在所难免，欢迎读者批评指正。

2003 年 7 月

第二版序

我对《Real-Time UML：Developing Efficient Object for Embedded Systems》第一版的成功深感欣慰。我认为第一版之所以受欢迎得益于实时和嵌入式系统开发中对象技术（一般来说）和 UML（具体来说）的时效性和适当性。在刚出第一版时，UML 就有成为面向对象系统开发主力军的迹象。然而，即便是它最坚定的支持者也为它在开发者中受欢迎的速度和程度感到惊讶。一位曾支持另外一种建模方法的方法学家对我说，“我忽视了 UML，所以很快就遇上了大麻烦。”不管从达尔文主义的角度来看，还是从它的技术优势来说，UML 都取得了巨大的成功，并成为对象领域内占统治地位的技术。

随着嵌入式系统变得越来越复杂，过去的那套老办法彻底不管用了。当前系统的复杂程度驱使开发人员从不同的视角来构建模型，以便理解并规划系统的各个层面。这些视角包括物理或者部署视角，以及逻辑或本质视角。二者都必须支持结构和行为两方面。这就说明了 UML 是什么，以及它为什么如此成功。

读者

本书面向在职的专业软件开发人员，以及计算机科学专业的大三和大四学生。本书也可用作大学本科以及研究生教材，但是本书偏向于实际开发，而不是理论性的介绍。在本书中几乎找不到什么方程式，但是在适当的地方都标出了引用，以方便读者找到它们更详细的理论和数学解释。阅读本书至少需要掌握一门编程语言，并且大概知道面向对象和实时系统的基本概念。

目标

本书和第一版的目标一样，仍然希望成为介绍 UML 以其表示法和语义在实时和嵌入式系统开发中的应用方面的一本好读物。除本书之外，市面上还有一本关于 UML 和实时系统的书：《Doing Hard Time：Developing Real-Time Systems with UML，Objects，Frameworks，and Patterns》（Addison-Wesley，1999），这也是我写的。《Doing Hard Time》对实时系统的讲述更加深入，重点放在对象可调度性的分析、行为模式在状态图模型构

建中的作用以及如何有效利用实时框架上。它更加深入地探讨了实时系统，并且正好也使用了 UML 来表达这些概念。与此形成鲜明对比的是，《Real-Time UML》主要讲的是 UML，其次才是如何利用 UML 捕获实时系统的需求、结构和行为。

除了第一版的最初目标，第二版又增加了另外两个：1)使本书与 UML 标准的最新变化保持一致；2)在第一版反馈的基础上增强本书的有效性。

自从 UML 被 OMG 采纳为标准以来，出现过两次修订。第一个修订版 1.2 几乎全是编辑性的，没有显著的改动。而 1.3 则在各方面都进行了大的改动。例如，用依赖关系的《includes》构造型替换用例泛化的《uses》构造型，从而改进了不少东西。

同样，UML 1.1 中动作的概念严重依赖于用来捕获其细节的“未解释文本”。UML 1.3 细化了该元模型，从而包含了更多种类的动作，同时使行为建模更加完整。动作语义元模型以及它如何与对象的消息发送相关，将在第 2 和 4 章讨论。

1.3 版中对状态图模型做了很多修改。本书的第一版在状态图上下了很大功夫，而这一版在利用状态图进行行为建模上花了更多的心血。其中大部分精力都放在了状态图新增的特性上——同步伪状态、桩状态等。因此，关于对象行为建模第 4 章做了很大的改动。

我在各个领域（从高级医疗成像到下一代智能、自动宇宙飞船）中进行咨询工作的最新体验，以及第一版读者的反馈，都在第二版中得到了反映。例如，无数次的咨询经历告诉我很多开发者在理解并利用用例来捕获实时和嵌入式系统需求方面有很大的困难。为了解决这一问题，我开发了一个叫做 Effective Use Cases 的一日课程，曾在 NASA（美国宇航局）以及别的地方讲过。那些已被证明在该领域内行之有效的准则在本书第 2 章得到了体现。同样，本书还提供了适用于捕获对象模型或者状态行为的技术和策略。

本书的另一个变化是产品开发中使用 UML 的有效过程的细化。我称此过程为嵌入式系统的快速面向对象过程（Rapid Object-Oriented Process for Embedded System, ROPES）。自从第一版面世以来，我被问得最多的问题就是关于在开发实时和嵌入式系统的项目组中如何成功地部署 UML。为此，第 1 章专门解释了这一过程，并标识了在迭代生命周期的不同阶段产生的工作活动和工件。事实上，本书组织的基础正是 ROPES 过程，贯穿第 2 到 7 章[1]。

UML 的目标是提供一个标准，但实际上却不尽人意，因为市场上的各个供应商都试图使自己的产品与众不同。时代的发展很自然地要求供应商们提供 UML 目前不能提供的、新的、潜在的、有价值的模型构造，因而有几个供应商开始声称他们的几项新特性将成为一些尚未宣布的实时 UML 的一部分。有意思的是，其中的有些供应商甚至连 OMG 都没有参与过，而其他的供应商则提供了互不兼容的“增强”。通过在开发人员社区内传

[1] 关于 ROPES 过程的更多信息可以在 I-Logix 的站点 www.ilogix.com 以及另一本书《Doing Hard Time: Developing Real-Time Systems with UML, Objects, Frameworks, and Patterns》（Addison-Wesley, 1999）上找到。

播这种心理恐怖战术——FUD 恐惧、不确定和怀疑，最先被 IBM 采用的一种行销伎俩），这些供应商极大地伤害了他们的顾客。开发人员应该知道采用这种单一来源的概念时，机遇以及风险是并存的。这些特性也许能够使系统的建模变得容易些（尽管在很多情况下，这些所谓的增强做不到这一点），但是也把开发人员死死地限定在一家公司的工具上。另一个风险是，当这些模型不再适合 UML 标准时不能在各个工具之间进行模型交换。这极大地减少了开发人员使用 UML 的好处。为了消除某些这样的 FUD，我专门增加了附录 B，以告诉人们更改标准意味着什么，为什么单个厂家不能声称拥有 UML 标准（它实际上属于 OMG），以及在未来的几年中 UML 可能会发生哪些变化。

最后，我建议感兴趣的读者访问 I-Logix 公司的官方网站，www.ilogix.com。那里可以找到很多相关主题的文章，可能是我写的，也可能是别人写的。另外，还可以找到 UML 规范、工具说明，以及相关站点的链接。

Bruce Powel Douglass 博士
1999 年春

关于作者

Bruce 由俄勒冈荒地的狼群抚养成人。3 岁时开始自学读书，不到 12 岁就开始学习微积分。14 岁辍学游历美国，几年后进入俄勒冈大学学习数学专业，并最终获得俄勒冈大学的运动生理学科学硕士学位、USD 医学院的神经生理学博士学位。他在医学院期间创立了一个名为自相关因子分析的数学分支，为研究多细胞生物神经系统中的信息处理提供了一个强有力的数学工具。

Bruce 作为软件开发人员，在实时系统领域工作了近 20 年，是实时嵌入式系统领域内著名的讲演者和作者。他是嵌入式系统（Embedded Systems）大会和 UML 世界（UML World）大会顾问委员会的成员之一，并在该会议上讲过关于软件估算和调度、项目管理、面向对象的分析和设计、通信协议、有限状态机、设计模式和安全临界系统设计方面的课程。在实时、面向对象的分析和设计中有多年的开发和授课经验。他还为实时领域内的很多杂志和期刊撰文。

Bruce 目前是实时系统开发工具的主流厂商 i-Logix 的技术总宣传师（Chief Evangelist）[1]，在 UML 规范的制订方面与 Rational 和其他 UML 合作伙伴有密切的合作。他是对象管理组（OMG）的实时分析和设计工作组的主席之一，负责 UML 未来可能的实时扩展。他还为构建大规模、实时、安全临界系统的很多公司提供咨询、培训和指导。他还写了其他四本软件方面的书，包括《Doing Hard Time: Developing Real-Time Systems with UML, Object, Frameworks, and Patterns》（Addison-Wesley, 1999），他甚至还写过乒乓球方面的短篇教材。

Bruce 喜欢古典音乐，古典吉他弹得很有专业水准。他参加过多场体育比赛，包括乒乓球、自行车比赛、赛跑以及身体完全接触的跆拳道，不过目前还只是与打不还手的静物交手。他与两个儿子目前正计划到冰天雪地的北方旅行。可以通过 bpd@ilogix.com 与他联系。

[1] 技术总宣传师类似于首席科学家。

目　　录

第 1 章
实时系统和对象简介

实时应用的大小和范围从手表和微波炉到工厂的自动化系统以及核电站的控制系统，变化很大。使用通用的开发方法来开发实时系统，意味着这种方法必须满足苛刻的性能要求以及 4 位和 8 位控制器的大小限制，并且需要适应强大的处理器的网络化阵列，以协调这些处理器的活动，使之达到通用的目的。面向对象的方法虽然不是什么银弹（silver bullet），却显著地改进了实时系统开发中传统的结构化方法。

所谓实时系统是指时效性对于保证正确性极其重要的系统。面向对象的建模方法天生就适于捕获在性能上具有硬时限（hard deadline）的系统的各种特征和需求。

讨论的表示法和概念

实时系统的特殊之处
关于时间
实时操作系统
对象的优点
对象和 UML
UML 表示法

1.1 实时系统的特殊之处

一些流行的计算机书籍往往给人这样一种印象：大多数计算机都放在桌面上（或者膝盖上），并且运行的是 Windows。从已部署系统的数量上来说，嵌入式的实时系统远比人们可以看到的桌面系统多。一个中等富裕程度的美国家庭可能有一台，甚至两台标准的台式计算机，但是却有更多的智能消费设备，每台设备中都包含一个或多个处理器。从洗衣机、微波炉、电话、立体声、电视到汽车，嵌入式计算机无处不在。它们能帮助

我们烤松饼，甚至还能辨别岳母打过来的电话。在工业中，嵌入式计算机用得更广。火车、交换系统、飞机、化学过程控制和核电站等都使用计算机来安全、有效地提高生产率并改善我们的生活质量（不用说，它们也为我们提供了大量的就业机会）。

构建嵌入式计算机的软件相对于桌面计算机来说要困难些。实时系统除了桌面应用程序遇到的问题外，还包括很多独有的问题。非实时系统不关心时效性、健壮性或者安全性——至少不像实时系统那样关心。实时系统通常没有传统上的显示器和键盘，它们位于那些显然是非计算机化的设备内部。这些设备的用户可能从来就不知道它们里面还有内置的计算机，更不知道系统是如何工作的。因此用户在内心深处并没有将这些设备看作是像计算机那样的设备，而是把它们当作提供服务的电气或机械设备。这样的系统必须能够在最恶劣的环境中连续运行数日甚至数年。它们提供的服务和控制也必须是自动和及时的。但同时，如果这些设备出现危险性故障，将造成巨大的危害。

实时系统中的设备有很多性能约束。硬时限是指必须无条件满足的性能需求。错误的处理以及系统故障构成了错过的时限（missed deadline）。在这种系统中，迟到的数据是坏数据。软实时系统具有以下时间约束特征：a）偶尔错过，b）错过一小段时间或c）偶尔完全跳过。软实时系统的另一个常见定义为：只受平均时间限制约束的系统（这样的例子包括联机数据库和机票预定系统），不过这样的约束实际上指的是吞吐量需求，而不是特定动作的时效性。在软实时系统中，迟到的数据仍可能是好数据。本文提供的方法可应用于所有性能受约束的系统的开发，不管是硬系统还是软系统。当我们只提到实时系统一词时，通常指的是硬实时系统。实际上，大多数实时系统都是硬实时和软实时约束，再加上一些没有时效性要求的需求混合而成的。尽管眼下硬实时约束占统治地位，但通用的做法还是分别对待这些不同的方面。

嵌入式系统将计算机作为大型系统的一部分，而不为用户提供标准的计算服务。桌面式PC不是嵌入式系统，除非它位于x线断层成像扫描器或者其他一些设备内。而计算机化的微波炉或者VCR却是嵌入式系统，因为它们不进行“标准的计算”。在上面两种情况中，嵌入式计算机都是一个大型系统的一部分，为用户提供非计算方面的特性，比如爆玉米花或者展现施瓦辛格将电话亭连根拔掉的壮观情形[1]。

大多数实时系统直接和电气设备交互，却不与机械设备直接相连。专为应用程序编写的自定义软件通常必须控制这些设备，实时系统程序员也因此而成为“裸机代码编写者”。你不可能仅通过一个标准的设备驱动程序或者Windows VxD来控制自定义的硬件组件。编写这些设备的驱动程序需要对设备进行低级的操纵。这种编程方法需要非常了

[1] 如果确实存在的话，应该是一个感人的突击队员的故事。

解设备的电气属性和定时特性。

实际上，所有的实时系统对硬件要么监控要么控制，或者两者兼而有之。传感器向系统提供了它外部环境的状态信息。医疗监控设备，比如心电图记录仪，就使用传感器来监控病人和仪器的状态。气流速度、引擎推力、姿态和高度传感器为正确地执行飞行控制计划提供了关于飞机的足够信息。直线和角度位置传感器能感知机器人手臂的位置，并通过 DC 或者步进电机来进行调整。

很多实时系统利用传动器来控制它们的外部环境，或者引导一些内部过程。飞行控制计算机可以控制引擎推力和双翼及尾翼的方向，以满足飞行参数的要求。化学过程控制系统能控制向混合容器中添加反应物的种类、数量以及时间。心脏起搏器通过连接在心室内壁（右边）的电线来使心脏按照适当的节奏跳动。

大部分包含传动器的系统很自然地也包含传感器。虽然存在一些开环控制系统[2]，但大部分的控制系统都使用环境反馈来保证控制环正常工作。

标准的计算机系统几乎完全面向用户，而不是其他任何东西[3]。而实时系统虽然也可能是面向用户的，但却更关心那些传感器和传动器。

由此产生的问题之一就是实时系统不会按照我们选择的时间和方式工作。外部事件通常是不可预测的。因此系统必须在事件发生时做出反应，而不是在自己方便时才来处理。ECG 监视器如果想有存在的价值，就必须在病人心脏停止跳动时，立即报警。系统不能等到自己的负载较少时，才发出警报。很多硬实时系统事实上是反应式（reactive）的，它们的响应必须严格地与外部事件同步。我们后面可以看到的控制环，对时间延迟很敏感。因此，如果系统的反应有延迟，就会破坏控制环的稳定性。

多数实时系统都会完成一件或一组高级任务。这些高级任务的实际执行过程需要很多同步的低级活动。这就是所谓的并发（concurrency）。由于单处理器系统一次只能干一件事情，所以它们采用调度策略（scheduling policy）来控制任务执行的时间。在多处理器系统中，由于处理器可以异步运行，所以可以获得真正的并发。这种系统中的单个处理器也可以伪并发地（pseudoconcurrently）调度线程（在任一时刻，只有一个线程在执行，但活动线程根据调度策略的不同而不同）。

嵌入式系统通常采用最便宜的计算机（当然功能就不太强），只要能满足功能和性能需求就行了。实时系统将硬件和软件作为一个完整的系统包，一起发售。由于很多产品对成本极其敏感，因此市场和销售上的考虑促使人们纷纷采用更小的处理器和更低的内

[2] 在开环系统中，被执行的动作的反馈不用来控制该动作。在闭环系统中，被监控的动作以及传感到的数据被用来修改该动作。

[3] 虽然桌面式计算机还必须在后台连接打印机、鼠标、键盘和网络，但这也是为了方便用户的使用。

存。小CPU和低内存可以降低生产成本。这种成本被称为平均经常成本（recurring cost），因为每生产出一件产品，它就变化一次。而软件没有很明显的平均经常成本——所有的成本都包含在开发、维护和支持活动中，因此看上去很随意[4]。这意味着，在降低硬件成本的同时，往往会增加软件的开发成本。

在UNIX下，如果开发者需要一个大的数组，他可以只为1000000个浮点数分配空间，并且不需要过多考虑结果。有谁会去关心程序是否用到了所有的空间呢？工作站有很多兆的RAM，并且有上G的硬盘存储形式的虚拟内存。嵌入式系统开发者不能做此简单的假设。他们必须做更多的事情，以避免复杂的算法和大量的性能优化。很自然地，这使得实时软件的开发和维护变得更加复杂和昂贵。

实时开发者经常使用PC和工作站上的工具，但他们的目标却是更小的、功能也更少的计算机平台。这就意味着他们必须使用跨编译器的工具，但是这些工具的性能通常没有广泛使用的桌面工具稳定。另外，目标平台上可用的硬件设施（比如定时器、A/D转换器和传感器）要想在工作站上模拟不那么容易。开发环境和目标环境之间的差异使得开发者执行和测试代码变得更费时费力。由于大多数小型目标平台上缺乏成熟的调试工具，测试也变得复杂起来。小型嵌入式目标通常连一个查看错误和诊断消息的显示器都没有。

实时开发者通常还必须为那些暂时还不存在的硬件设计和编写软件。这就给开发者带来了真正的挑战，因为他们不能确认这些硬件的功能。集成和验证测试也因此变得更加困难和费时。

嵌入式实时系统必须经常连续运行一段时间。如果飞到了纽华克上空时由于GPF[5]的原因，不得不重置飞行控制计算机，那将是一件非常尴尬的事情。这同样适用于心脏起搏器，它植入后通常可以用到10年。无人太空探测器也必须在核能或者太阳能电源的作用下，正常运行数年。这和桌面计算机不太一样，后者每天至少都得重置一次。当发现一个隐藏的Excel“特性”时，重启桌面PC是可以接受的，但对于维持生命系统的呼吸机和商务喷气式客机的控制仪表来说，就万万不可。

嵌入式系统环境通常是不利的，且是计算机不友好的（computer-hostile）。在外科手术中，外科手术仪器产生的电弧对病人的伤口进行灼烧式切割。这将产生极高的EMI（电磁干扰），因此给那些没有采取保护措施的计算机设备造成物理伤害。即使这些伤害不是

[4] 不幸的是，很多公司往往只关心降低硬件的平均经常成本，而没有考虑所有的开发成本。关于这个话题，完全可以写出另一本书，我们这里不作讨论。

[5] General Protection Fault（一般性保护错误），微软的Windows 3.1版本发布以后人们常遇到的一个术语。