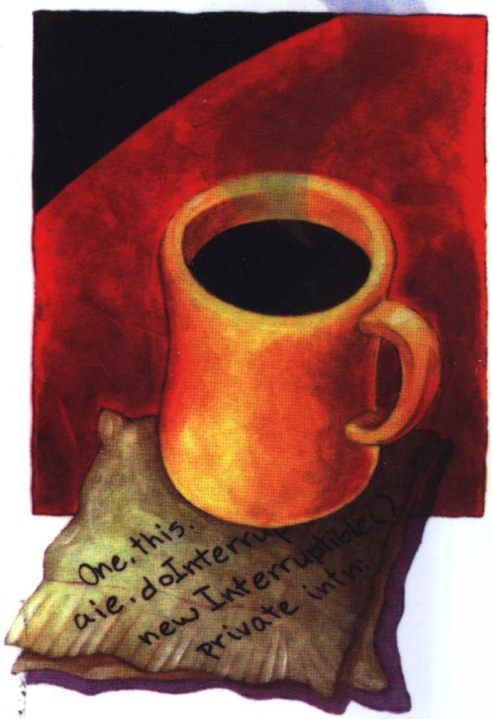
Sun公司核心技术丛书

HZ BOOKS
PH PTR

实时Java平台编程

Real-time Java Platform Programming



- 实时Java规范的权威参考资料——由规范的制订者之一编著
- 全面介绍Java 实时开发——无需实时经验

(美) Peter C. Dibble 著
滕启明 等译

Sun microsystems

机械工业出版社
China Machine Press

Sun 公司核心技术丛书

实时 Java 平台编程

（美） Peter C. Dibble 著

滕启明 等译

机械工业出版社
China Machine Press

本书是一本介绍实时 Java 平台编程的参考书，尽管本书主要是针对 RTSJ 平台的，但绝不仅仅是 RTSJ 的参考手册。作者从语言闭包、高解析度时间、异步事件、实时线程、领域内存、不朽内存、物理内存等角度介绍了 RTSJ 平台对实时编程所提供的支持。在介绍每项功能的同时，作者还给出了大量的代码。本书中涉及到的技术、观念广泛、新颖。由于本书的作者亲自参与了 RTSJ 规范的编写，并且是 RTSJ 专家组的成员，所以书中的很多见解及提示信息都有其独到之处，很值得学习。另外，书本提供的大量代码都经过了测试，具有很高的实用价值。

本书适用于大专院校计算机专业高年级本科生和研究生。对于从事软件开发、培训工作的管理人员，以及从事嵌入式软件开发、实时系统研制、Java 应用编写的开发人员也有重要的参考价值。

本书版权登记号：图字：01-2002-4815

图书在版编目(CIP)数据

实时 Java 平台编程/（美）迪博（Dibble, P.C.）著；滕启明等译. -北京：机械工业出版社，2003.1
（Sun 公司核心技术丛书）
书名原文：Real-time Java Platform Programming
ISBN 7-111-11583-X

Ⅰ. 实… Ⅱ. ①迪… ②滕… Ⅲ. JAVA 语言—程序设计 Ⅳ. TP312

中国版本图书馆 CIP 数据核字（2003）第 006340 号

机械工业出版社（北京市西城区百万庄大街 22 号 邮政编码 100037）
责任编辑：周 睿
北京瑞德印刷有限公司印刷·新华书店北京发行所发行
2003 年 3 月第 1 版第 1 次印刷
787mm × 1092mm 1/16 · 17 印张
印数：0 001 - 5 000 册
定价：35.00 元

译 者 序

20世纪90年代以来，随着计算机网络技术的发展，特别是网络应用的大量普及，进一步推动了相关领域的发展，带动了国民经济，潜移默化地改变着人们的工作、学习、生活方式。移动计算、普适计算、信息家电、无线通信等领域随着硬件技术、通信技术、软件技术的发展而得到长足的进步，并在近年来引起了业界的广泛关注。

实时计算作为一个研究领域已经存在了多年，但很多研究人员、开发人员甚至对实时系统与嵌入式系统都不予严格区分。嵌入式系统的开发通常采用完全定制的模式，所有代码都要从头编写。之所以存在这样的状况，一方面可能是因为应用领域、性能需求方面的限制，另一方面则往往是因为没有可以复用的代码。对处于底层的系统软件进行复用，其难度要比对上层软件的复用大得多。其原因在于底层软件往往是与硬件相关的，而且其性能对于整个系统而言相当关键。所以很多嵌入式系统、实时系统一般采用汇编语言或者C语言编写。

作为计算机编程语言的后起之秀，Java具有众多很有吸引力的特性：可移植性、安全性、完全面向对象、强大的类库支持等等。随着Java语言在应用系统开发中的广泛使用，已经有人开始考虑把Java语言应用到实时系统软件开发中，并且也确实有人这样做了。本书即是介绍实时Java平台的一本参考资料。

本书的组织方式比较特别：从第1章到第7章几乎很少提及RTSJ，而主要关注Java平台本身（如Java虚拟机的体系结构、垃圾收集），以及与实时计算相关的基本原理（如硬件体系结构、实时调度原理）；从第8章开始才真正开始对RTSJ平台做深入细致的介绍。尽管本书对实时Java平台编程的介绍主要是针对RTSJ平台，但本书绝不是RTSJ的参考手册。作者从语言闭包、高解析度时间、异步事件、实时线程、领域内存、不朽内存、物理内存、同步等各个角度介绍了RTSJ平台对实时编程所提供的支持。在介绍每一项功能支持的同时，作者还给出了大量的范例代码。

同时，由于作者Peter C. Dibble亲自参与了RTSJ规范的编写，并且是RTSJ专家组的成员，所以书中的很多见解，甚至一些随时想到的提示信息都有其独到之处。这也会是读者在阅读此书时的收获之一。

在本书原作者编写此书时，RTSJ规范尚未正式发布，更谈不上成熟。英文原书是随RTSJ规范一起出台的，同时出台的还有一份参考实现。作为本书的译者，一方面力求能够用中文准确地再现原作者的思想，同时也要对一些专业词汇进行斟酌。在出现专业术语时尽量注明其英文原文，以避免读者产生混淆。但是由于中文和英文的表达方式不同，很难做到兼顾中文可读性和英文原文的韵味，这一点敬请读者谅解。

总体而言，此书中涉及到的技术、观念广泛、新颖。适合于大专院校计算机专业高年级本科生和研究生参考。对于从事软件开发、培训工作的管理人员，以及从事嵌入式软件开发、实时系统研制、Java应用编写的开发人员也有很重要的实用参考价值。

参与本书翻译、审校工作的还有北京大学计算机系的顿楠、胡建均、余啸海、朱伟等。北京大学计算机系陈向群教授参与了本书的翻译、审校工作，并在整个翻译过程中给予了大力支持。在此一并表示译者的诚挚谢意。

限于译者的水平，译文中难免存在纰漏不足之处，欢迎广大读者不吝指正。

滕启明

2002 年 10 月于北京大学

作 者 序

——关于本书的写作过程

实时计算（Real-time computing）——带期限的计算（computing with deadlines）——是涉及到所有程序员的，但几乎未受到认真关注的领域。控制飞旋的锯条的机器是一个实时问题；承诺在两秒钟内响应查询的 Web 站点也是；为让用户感觉舒适而要对每次击键在十分之一秒内做出反应的文本编辑器也是。从广义来说，两周一次的工资表编制也算是一个实时系统。

对一般程序员而言，他们会对期限问题感到头疼，从而会在高性能算法、编译器优化、快速处理器等方面投入很多精力。尽管这些研究很重要，不过他们还是忽略了与恒定的性能有关的问题。导致时间性不可靠的因素有很多，而实时编程（Real-time programming）的任务就是要解决这些问题。

在我编写此书的这些年里，最快的 Java 虚拟机（Java Virtual Machine）已被充分地改进，进而使得 Java 平台的性能可以赶上甚至超过C++ 的性能，不过这些应该是另外一本书的内容。当然，实时程序员会对完成一项计算所花费的时间感兴趣，不过更重要的问题是计算是否总能按期完成。对于那些可能造成时间延误的所有因素，是否都能给予合理的解释？

实时 Java 规范（Real Time Specification for Java，RTSJ）把重点放在了与系统有关的因素上，这些因素与必须被满足的期限有关，主要是时间本身和可能导致不可预料的延迟的事物。本书将把讨论的焦点放在这些问题上。

在此，我向 Java 实时专家组的其他成员致谢。有六位“专家”是值得特别致谢的，他们是 Greg Bollella、Ben Brosgol、Steve Furr、James Gosling[㊀]、David Hardin 和 Mark Turnbull。没有你们就没有这个规范，也就不会有这本书，我也就错过了至今为止可能是最激动人心的一次规范编写活动。

我觉得本书的整个写作过程中还有一点特别有意思。我们都被当作公司的代表派出[㊁]。不过，在任何情况下，一旦我们工作起来，就好像我们的公司命令我们要忽略任何商业因素开发一个好的规范一样。或许是 Sun 公司的错误，他们把 James Gosling 派来。他不仅技能高超、在 Java 界倍受尊敬，他还是一个不拘细节的、正直的科学家。或许我们可以进一步归功于 IBM，是 IBM 确定了把我们选出来的基本标准。不管怎样，这种做法的确不错。所有的规范都应该以这种方式产生！

参考实现小组的主要成员也值得特别提一下。Doug Locke、Pratik Solanki 和 Scott Robbins 编

㊀ James Gosling 是 Java 语言的发明者。——译者注

㊁ 负责 RTSJ 开发的小组来自不同的公司——Peter Haggar 来自 IBM，任组长：Greg Bollella 来自 Sun 公司；Paul Bowman 来自 Cyberonics 公司；Ben Brosgol 来自 Anoix 公司；Peter Dibble，即本书的作者来自 Microware 公司；Steve Furr 来自 QNX 公司；David Hardin 来自 aJile 公司；Mark Turnbull 来自 Nortel 公司；James Gosling 来自 Sun 公司。

写了大量的代码，并在规范开发的最后一年期间参与了专家组会议的召集工作。他们不但用代码的形式让规范变得生动，还帮助我们重新设计了其中的某些设施。

我在规范尚未完成以前就开始编写本书了。原来的目标是在规范成为最终版本的时候能够有一套书出来。一本规范、一本参考实现以及一些关于“怎样使用”的书应该同时提交。这项工作一开始就需要很大的耐心，可是事情的发展并不总是如所想像的那样。

在最初的规范发布之前，规范的发展还是条理清晰的。在后来进行的工作中，我们遇到了一些困难。规范中最为复杂的部分发生了重大的变化，原因归结为参考实现的编写以及我在写作此书时的一些感受。专家组成员一直对我们的设计的可实现性不敢予以肯定，这些设计包括领域内存管理（Scoped memory management）、物理内存访问、控制的异步传输（Asynchronous transfer of control）等。我们达成的一致意见是：把一些相当大胆的设计放进初始规范中，然后看看参考实现小组会怎样处置它们。结果表明我们必须对控制的异步传输规则进行一定的约束，还要为领域内存增加很多限制。后来我们发现领域内存和线程之间的交互使得在无堆内存（Non-heap memory）中启动线程相当困难，这样做甚至一无是处。后来我们修正了这个问题，同时我还对某些章节写了再改、改了再写。

每个在得到广泛应用前发布的规范，都需要一个作者来尽可能地解释该规范，并编写利用该规范的代码。这位作者将会发现实现人员和测试软件包工具都没有注意到的问题，不过作者自身最好喜欢该规范。如果某项功能特性存在问题，那么关于这项特性的章节常常会变得很长、很复杂，因为它要阐述蕴含在设计背后的功能和原理。最终，得到的结论会是“这项特性并不复杂也不够精彩，它完全是错误的!”于是这一章就成了垃圾，新的一章出现了，解释的是新的设计。

很高兴专家组和参考实现小组都给予了我相当的支持。最终的规范在 2001 年晚些时候提交给了 www.rtj.org，参考实现也在 2002 年早期提交到这个网站。而这本“如何使用”方面的书也面市了。我们做得还不错吧!

Peter C.Dibble

前　言

本书分为二大部分。第1章到第7章讲述RTSJ的背景知识，剩下的部分是关于RTSJ的内容。如果你已经掌握了实时调度或者并不关心调度问题而想直接阅读代码，你可以从第8章开始往后阅读。尽管跳过前面7章是可以的，但我不推荐这样跳来跳去地阅读。几乎没有哪章可以自成一体。在读完本书后，你还可把它作为参考资料用，所以我建议你还是从头开始、按顺序地读一下本书。

本书可以与RTSJ规范、参考实现配合使用。你可以在www.phptr.com/dibble或者www.rtj.org上找到规范和参考实现。初始的RTSJ文档是Addison-Wesley Java丛书的一部分。不过，RTSJ的初始版本已经被最终版本——1.0版本所取代。目前最新的RTSJ版本只能通过可下载的PDF和HTML格式得到。

参考实现是RTSJ for Linux的一个完整的、可用的实现。本书中提到的例子几乎都在参考实现上测试过。我曾在运行有Red Hat Linux和TimeSys Linux的PC上使用过该参考实现，在X86 Linux的其他版本上也应该没有问题。不过参考实现依赖于底层的操作系统进行调度，所以你会发现类似于优先级逆转避免（Priority Inversion Avoidance）这样的功能特性会取决于你所使用的Linux版本。

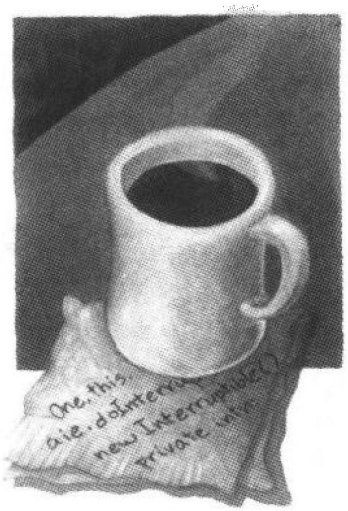
参考实现的源代码是可以获取的。其中一些是从Sun CVM[⊖]派生而来，可以在Sun社区源码授权协议（Sun community source license）下获得。参考实现中与Sun的代码没有关系的那部分受开放源码协议（Open source license）的约束比较小。

虽然参考实现用于实验相当不错，但由于它不是为商业使用而设计的，它没有在性能或内存使用方面加以注意，而这些或许正是你期望从商业产品中获得的。

在www.phptr.com/dibble上，你可以找到重要的Web站点的链接、本书的修正和补充、以及其他诸如源代码等有用的东西。

⊖ CVM是Sun公司开发的Java虚拟机（JVM）之一，英文全称是C Virtual Machine，专为需要Java2虚拟机特性集合而又要求占用空间小的设备而设计，感兴趣的读者可参考java.sun.com/products/cdc/cvm/。——译者注

目　录



第1章 概 貌

本章内容概要：

- Java技术和实时
- 实时的定义
- Java的问题
- 实时Java的问题
- 总结

对实时编程而言，选择Java平台看起来有点异乎寻常：垃圾收集随时都会让系统停止运行，这使得Java技术的时间特性很糟糕。取决于程序本身和Java平台的细节，Java技术的性能会比用C++编写的同样程序慢上1到30倍。要不是实时界的人们觉得没有更好的工具来代替它，Java平台就会立刻遭到拒绝。

Java技术的好处相当迷人，在某些实时系统中曾经使用过标准的Java平台，而其展现出的前景也证明应该设计和构建Java技术的实时扩展——RTJava平台。

1.1 Java技术和实时

嵌入式实时系统的程序员是实时界最为主要的成员，不过几乎每个程序员都会偶尔地要处理一些实时问题：每个编写用户界面的人都关心恒定的响应时间；电信设备具有规定的限期；火车、飞机、电梯、卡车、送货系统中的包裹；对电视编程；网络中流动的数据包；等待校车的孩子——所有这些系统都包含时间约束，并且都（除了让孩子按时赶上校车）可能涉及到计算机。

实时编程和其他类型的编程类似，不过要难一些。与一般的程序一样，实时程序必须产生正确的结果；并且必须在适当的时间产生结果。传统观念认为，实时编程一般采用过时的[⊖]或者晦涩的[⊜]语言。实时Java却为实时程序员提供了一种为生产率而设计的、现代的主流语言。

有些人批评Java平台的设计人员为换取生产率而牺牲了性能，这些人必须面对两个事实：

1）编译器应该能够对Java面向程序员的、友好的特性所带来的开销做出优化，并使得尽可能减少这部分开销。

2）摩尔定律（Moore's law）表明处理器的能力得到了迅速提高，Java引入的所有合理的、恒定因素带来的额外开销都会逐渐被处理器的增强所弥补。

如果你对性能问题过于敏感，那就去使用C或者汇编语言吧，不过在对C应用编码之前

⊖ 汇编语言本身可能不算古董，不过采用汇编语言来编程则因RISC处理器和复杂的优化编译器而遭摒弃。

⊜ FORTH是一种相对受欢迎的晦涩语言。美国国防部还存有一箩筐用于实时系统的其他晦涩语言。

应该先设计、编写、调试一个 Java 应用。实时 Java 就是针对相同的主题设计的。

注意　对本书而言，我们约定，Java 平台缓慢且存在垃圾收集延迟。这些问题并不是一成不变的。在 Java 虚拟机（Java Virtual Machine，JVM）中采用的技巧使得执行变快、垃圾收集更有礼节，不过这些解决方案不是本书的重点。

实时并不一定意味着“真正的快速(real fast)”。一根针飞速插入价值 200 元一个的玻璃试管，并要在还差 1 毫米就要撞到试管底部时突然停下，对于控制这样一根针运动的计算机而言，计算速度不是问题。如果系统慢，解决方案众所周知：分析性能、改进算法、调整代码、升级硬件。而解决不稳定的行为问题则要难得多：只能通过查找并去除问题的根本原因来修正，而根据定义，这些原因只是零散冗现。提高性能直到问题消失的做法往往弊大于利。系统或许在测试的时候不再出错，但在发布以后仍有可能出错。

1.1.1　实时编程需求

实时程序员要求可预测性（predictability）：假如程序在测试的时候能及时地把针停住，那么这根针必定从来不会撞到试管，除非硬件出问题。最糟糕的调试问题出现在软件几乎每次都运行正常的情况下。要在测试的时候去除时间性错误，可靠地让这些错误永远不再发生，这只能是痴人说梦。要说服你自己，从本质上讲，能工作的修补品只能算是一道练习题——从一个有缺点的软件中把缺点去掉是办不到的，进行了修正后，错误没有再发生。但缺陷就肯定已被去除了么？

实时程序员也是在通常的工程环境中工作。软件设计必须针对很多目标做出优化：正确性、低开销、能快速投放市场、必备的功能特性等等。这些实时程序员所关心的不仅仅是实时方面。可预测性有助于快速地获得一个正确的实现，对软件的正确性和投放市场的时间也有帮助；不过开销也是一个问题：高速度和低内存占用（footprint）实际上就是效率。这两点有助于减少开销并且便于将来对功能特性集的扩充。如果某种工具或技术可以使软件运行得更快，那么它可能采用了以下几种技术方案中的一个：

- 让系统使用一个低性能的（更便宜一点）的处理器
- 把某些操作从专用硬件中转移到软件中
- 释放一些处理器带宽来提供其他的功能特性

在更快的处理器或者专用硬件的成本令人望而却步的前提下，现实的选择就是要么对软件进行优化，要么放弃计划。成本高昂这种说法是模棱两可的：在某些领域，花费多年的时间来实施一项工程，目的是降低硬件成本，即便是每单元几分钱，这样做也是明智的；而在其他领域，为了节省每个软件项目的工程时间而耗费成百上千美元添置硬件也是很平常的。

Java 有多大

基于在 Microware 上的实现，JDK 风格的运行时环境所需的最小 RAM 和 ROM 需求均超过了 16 兆字节的容量。

带有 e-mail 客户端、地址簿和其他类似应用的 PersonalJava 系统可以在 4 兆字节的 ROM 和 4 兆字节的 RAM 下运行。对于应用程序、类库、JVM、操作系统和支持构件，这些内存足够了。

EmbeddedJava 的运行时环境可以配置得比 PersonalJava 大很多或者小很多，不过它的下限很难低于 0.5 兆字节的 ROM 和 0.5 兆字节的 RAM。

KJava 虚拟机承诺能够运行在少于 128KB 的环境中。

RTJava 没有针对 Java 的性能做任何工作。如果有的话，RTJava 比普通的 Java 要慢一点，而普通的（解释式的）Java 要比 C 慢。（参看在 1997 年 1 月《Internet Computing》上 Arthur van Hoff 的《The Case for Java as a Programming Language》；以及 2000 年《IEEE Computer》上 Lutz Prechelt 发表的《An Empirical Comparison of Seven Programming Languages》）就目前而言，必须接受这种缺点：Java 比其他可选语言要慢，并且运行一个很小的程序所需要的内存大小都令人吃惊。

1.1.2 Java 和嵌入式实时

Java 不会很快把 C 赶出传统的嵌入市场。嵌入式程序员像猫一样保守，这种小心谨慎具有某种传统意味。嵌入式系统中的缺陷能够造成场面宏大的灾难性后果，而用来更新嵌入式软件的代价通常也很高。

采用某种技术，而这种技术尚未在若干已发布的系统中得到彻底检验，这样做嵌入式程序员会感到不安。采纳新技术的障碍受弱势通道效应（minor tunneling effect）影响。数量很少的、有冒险精神的团体会在嵌入式实时系统中尝试采用 Java，他们会谈论有关结果。如果实时 Java 被证明是一种好的嵌入式实时工具，它将在 5 到 10 年内成为一种通用的嵌入式编程工具。

1.2 实时的定义

实时问题把时间性（timeliness[⊖]，“在合适的、恰好的时间发生”）看做是正确性的评判标准。如果过程结束得晚了，就是错误的，或者至少比起那些按时完成的过程来讲不是那么让人满意。

时间性通常很重要。几乎没人会把字处理程序、编译器、工资单系统称为实时系统，不过或许他们应该这样做。如果一个字处理程序花了十分之几秒来回显一个字符，程序的使用者就会怀疑编程人员的能力问题了。如果一个编译器编译一个程序花费的时间比预想的长很多，其用户就会担心编译器是不是已经卡壳了。对工资单来说，时间性尤为重要。滞后的工资单会让等米下锅的人们感到恐慌和愤怒。

实时问题的范畴很大、而且多样化，不过有一些标准工具和技术适用于所有问题：针对可

⊖ timeliness 在中文中解释为“时间性”、“及时”、“好时机”。作者在原注中给出有关 timeliness 的定义取自《The American Heritage Dictionary of the English Language》，Third Edition，Houghton Mifflin Company，1992。英文原文中对这个词的解释是：Occuring at a suitable or opportune time。这里约定，在本书中这个词统一译成“时间性”。——译者注

预测性做出了优化的工具、对时间性（而不是吞吐量）进行优化的调度程序以及考虑到时间约束的分析技术。

实时问题的空间至少要有三个有用的度量：用以度量时间的精度、稳定性的比重、围绕最终期限的效用曲线。

1.2.1　测量的精度

在实时系统设计中用来度量时间的单位精度有助于刻划系统特征（参看图 1-1）。一个系统把每秒钟当成 1 秒、1000 毫秒、还是 10 亿纳秒？

1．*亚微秒*（Submicrosecond）

某些计算机问题采用以小于一微秒（百万分之一秒）的单位度量时间来表述。专用于某项重复性工作的通用处理器、或者特殊用途的硬件可以达到这个精度级别。将来我们或许能够在一个通用系统上处理这类问题，但现在还不行（为什么在不对环境进行严格控制的前提下，预测几个指令的执行时间会如此困难呢？原因可参看第 3 章）。

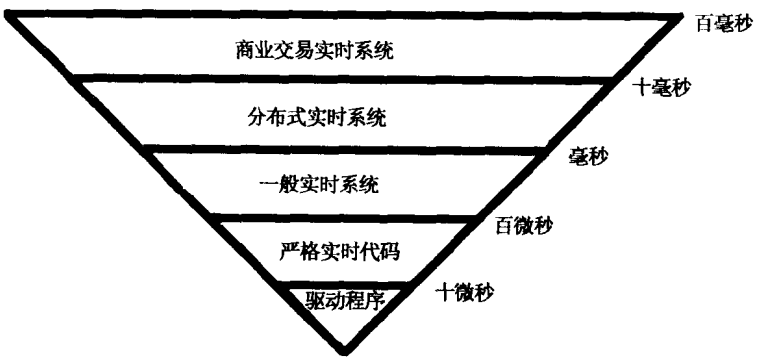


图 1-1　时间尺度金字塔

2．*十微秒*（Ten Microsecond）

软件常常处理以十微秒表述的规范，不过为这个精度的规范编写代码需要对下层的硬件有细致、深入的了解。这个精度层次的编码常常出现在精心编制的设备驱动程序代码中。与一般规则不同的是，其目标常常是在一个精确、短暂的间隔内完成计算。

3．*毫秒*（Millisecond）

普通实时软件常常处理以毫秒为度量的时间。大多数知名的实时系统都属于这个范围。这些系统可以用一般的工具进行编程，只要程序员对硬件、编译器以及系统软件的人为时间因素给予适当的注意就行了。

4．*百分之几秒*（Hundredths of a second）

分布式实时编程以百分之几秒来看待时间。时间的原子计量单位按网络通信进行，而网络环境是动态的。如果你无法容忍毫秒级别很不稳定的计时方式，就不要将应用程序中的那部分做分布式处理。

5．十分之几秒（Tenths of a second）

与人打交道的程序按十分之几秒的精度来看待时间，这是命令和响应时间的级别。当用户敲击键盘、点击鼠标、按下按钮、穿过光电检测器、触摸触摸屏、或者以其他方式向计算机下达指令的时候，程序都是以规范的响应时间精度来响应的。编写这些系统常常根本不考虑实时规范。性能问题是由更快的硬件或者一般的性能测试、调整方法来保证的。就实时的意义来说，这些系统在负荷较重的情况下总是“失效”。

1.2.2 一致性

我们所研究的问题是与计算机相关的问题。尽管在亚原子层次上存在随机性，电气工程师们已经在计算机出厂前排除了绝大部分的不可预测行为。任何遗留下来的随机行为都算硬件的错误。如果我们假定硬件没有问题，那么所有事物都是可预测的了。然而，为影响执行时间的所有因素找出依据太困难了，在很多处理器和软件平台系统上，执行时间所带来的问题跟不可预测没差多少。

在实时领域，术语确定论（determinism）的含义是在无须超人般努力的前提下，在问题所要求的精度级别，计时是可预测的。确定论是好事情。实时系统的设计过程中的工作包括为事件、对事件的响应制作时间表。如果没有确定性的（deterministic）操作系统和处理器，分析人员基本上无法预测某个事件会不会在它的最后期限之前到达事件的处理程序，更不用提事件处理程序是否会在最后期限之前完成计算了。

具有一致性比起单纯地具有确定性要好些。知道一桩紧急事件会在某一时刻到达你的程序，并且知道这个未来时刻将介于 10 微秒和 200 微秒之间，这很管用。若是能知道时间间隔将介于 50 到 70 微秒那更好。实时系统的设计可以针对所有确定性的环境，可是这样做必须假定系统将一直表现出其最可能坏的性能（Worst Possible Performance）。根据这种假定做设计不够划算，因为典型的时间都是接近最好情况的。具有一致性的系统能够将期望性能（Expected Performance）与最可能坏的性能之间的差距缩小，理想的方式是改进最坏情况的性能，而不仅仅是对典型性能进行调整。

稳定性要以性能为代价。要使得最佳情形（系统可以运行的最快速度）和最坏情形（最差的可能）性能尽可能接近，系统不能利用直觉信息或者启发式方法，也不能依赖“80/20”规则。一棵动态生成的二叉树可在线性搜索时间内退化成一个结构体；快速排序算法也可能需要 $O(n^2)$ 时间；直觉得来的信息也可能是错误的——这类错误将迫使程序检查该信息，并执行低效运行策略。解决这些问题的相应实时方法如下：采用自平衡二叉树；采用不同的排序算法（合并排序总体上比快速排序慢，不过是可预测的）；不要利用直觉信息。这样做所得到的软件的典型性能至少比那些在设计上针对典型性能而进行了优化的系统差 15%，不过其最坏情形性能可能会比传统设计好上几个数量级。

1.2.3 效用函数曲线图

如果实时系统延迟了会怎样？它错过了最后期限……然后呢？这个问题的答案决定了在从