HZ BOOKS
华章科技

ELSEVIER

华章专业开发者丛书

嵌入式与实时系统开发大师力作
C嵌入式工程按时交付的制胜法宝

C嵌入式编程设计模式

Design Patterns for Embedded Systems in C

An Embedded Software Engineering Toolkit

（美）Bruce Powel Douglass 著

刘旭东 译

机械工业出版社
China Machine Press

华章专业开发者丛书

C嵌入式编程设计模式

Design Patterns for Embedded Systems in C

An Embedded Software Engineering Toolkit

（美）Bruce Powel Douglass 著
刘旭东 译

机械工业出版社
China Machine Press

本书以面向对象的视角，重新审视嵌入式系统，全面总结了嵌入式系统中常见的以及关键的设计模式。本书提出了很多新颖的设计模式，为使用C语言编程的嵌入式系统开发者提供了强有力的工具。通过这些模式，开发者可以用最短的时间设计出性能好、稳定性强、安全性高的嵌入式系统或软件。本书针对嵌入式系统中从内存访问到事件调度，从状态机设计到安全性可靠性保证，对系统的设计以及性能表现的方方面面进行了详细阐述。

全书采用UML图形化解释，直观清晰；所有实例配有C代码实现，方便使用。本书适合面向专业软件开发人员和计算机专业的学生阅读。

Design Patterns for Embedded Systems in C: An Embedded Software Engineering Toolkit
Bruce Powel Douglass
ISBN: 978-1-85617-707-8

ISBN: 978-981-272-927-9

本书版权登记号：图字：01-2011-2874

图书在版编目（CIP）数据

C嵌入式编程设计模式／（美）道格拉斯（Douglass, B. P.）著；刘旭东译. —北京：机械工业出版社，2012.4
（华章专业开发者丛书）
书名原文：Design Patterns for Embedded Systems in C: An Embedded Software Engineering Toolkit

ISBN 978-7-111-37592-0

Ⅰ. C…　Ⅱ. ①道…　②刘…　Ⅲ. C语言－程序设计　Ⅳ. TP312

中国版本图书馆CIP数据核字（2012）第033876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：秦　健
北京市荣盛彩色印刷有限公司印刷
2012年4月第1版第1次印刷
186mm×240mm · 22.25印张
标准书号：ISBN 978-7-111-37592-0
定价：69.00元

凡购本书，如有缺页、倒页、脱页，由本社发行部调换
客服热线：（010）88378991；88361066
购书热线：（010）68326294；88379649；68995259
投稿热线：（010）88379604
读者信箱：hzjsj@hzbook.com

译 者 序

随着电子技术、通信技术等的飞速发展，嵌入式系统已经广泛地应用在工业控制、通信、航空航天、消费电子产品等领域，其所带来的效益不可估量。随着时间推移，嵌入式系统的需求量呈现指数增长，并且应用范围不断扩大，同时对系统的复杂性、稳定性、安全性以及关键性的要求也日益提高。嵌入式系统如何满足这种需求，怎样提高嵌入式软件的生产率，怎样以最短的时间开发出最令人满意的、高效可靠的嵌入式软件成为了摆在人们面前的问题。

本书以面向对象的视角，重新审视嵌入式系统，全面总结了嵌入式系统中常见的以及关键的设计模式。这些模式广泛应用于嵌入式系统或嵌入式软件中。本书还提出了很多新颖的设计模式，为使用C语言编程的嵌入式系统开发者提供了强有力的工具。通过这些模式，开发者可以用最短的时间设计出性能好、稳定性强、安全性高的嵌入式系统或软件，而且也能为系统日后的升级维护打下坚实的设计基础。读者能够从本书中系统地掌握嵌入式系统的设计模式，使用C语言以面向对象的视角设计系统、开发系统。本书针对嵌入式系统中从内存访问到事件调度，从状态机设计到安全性、可靠性保证，对系统设计以及性能表现的方方面面进行了详细阐述，也提出了很好的设计规则。

本书的作者拥有30年的嵌入式系统设计和开发经验，本书是他对嵌入式系统设计模式的一次详细的总结。本书的组织条理清晰，不仅是一本关于嵌入式系统设计模式的优秀书籍，更是一个针对嵌入式软件工程的工具箱。读者可以从这个工具箱中找到应用到工作中的经典模式，通过在开发设计过程中使用这些模式能极大地提高嵌入式系统或软件的功能和稳定性。此外，所有的模式都使用UML来图形化解释，读者可以更直观地了解模式，并且配有详细的C代码实现，极大地方便了工作中的使用。可以说本书是一本不可多得的嵌入式系统设计方面的好书。

本书在翻译的过程中得到了很多人的帮助和鼓励，在此感谢机械工业出版社编辑在本书翻译过程中给予的帮助，还要感谢Kourosh Farrokhzadi对本书部分内容理解方面给予的指导。由于时间关系，虽然尽最大的努力翻译，但是译文中难免有疏漏和错误之处，恳请读者批评指正。

前　言

嵌入式系统开发中占主导地位的编程语言显然是C语言，其他编程语言当然也有它们的魅力，但是所有嵌入式系统中80%以上是使用这种经典的编程语言开发的。行业的发展趋势是采用面向对象编程语言、Web客户端技术，而这些技术在嵌入式系统的裸机开发环境中不能实现或是由于内存和资源密集而不能有效部署。

设计模式是这些发展趋势之一。设计模式是对一种反复出现的问题的广义解决方案。设计模式有一些优点。首先，它允许用户以一种更为抽象的方式思考解决方案并且归纳它们重要的属性。由于所有的设计模式都是优化某些设计准则，而牺牲其他准则，不同的设计模式可用于相同的开发环境，但具有不同的收益和成本。通过将设计模式具体化为基本概念，我们能找到最好的方法优化系统和技术，并且找出途径来实现这个目标。

其次，设计模式允许我们重用那些已经证明在其他相似环境中有效的解决方案，这当然是比重用几行代码或个别函数更大范围的重用。因为设计模式可以分析其性能和优化性能，可以为特殊的问题选用最好的设计模式。

此外，设计模式给我们带来更大一组可重用的构建模块，用以描述我们的系统。如果你说系统使用“RMS多任务调度和三重容错方法的对称部署模式”，这概括了你的系统架构优化决策的重要部分。设计模式使我们有更多标准术语来表达设计内容，且更具表现力。

最后，设计模式提供了一种参考。如果你用Google搜索“设计模式”，你会得到超过3 000 000条结果。如果你搜索亚马逊在线书店，你会得到包含793本书的清单。在定义和使用设计模式领域有大量的工作在做，所以我们有丰富的可重用和可靠的解决方案，从中可以选择、组合并且应用适当的解决方案。

本书将设计模式的力量带给嵌入式C语言开发者。在适当的地方，我们使用基于对象的实现策略，通过代码实例来观察模式是如何实现和使用的。嵌入式开发中的设计模式分为如下几个方面：

- 硬件访问
- 并发
- 状态机实现
- 安全性和稳定性

每个方面都提供不同的模式解决共性的问题。

读者对象

本书面向专业软件开发人员和计算机专业的学生。通过求解例子中的实际问题，注重展示实际经验。本书假定读者有一定的C语言编程经验。本书不仅使用UML以图形化的方式表现结构和模式，而且用代码实例来清楚地展现设计和实现。

目标

本书的目标是为嵌入式 C 开发者提供一个工具箱，并且介绍如何恰当地使用这些工具。第 2 章讨论了称作 Harmony Process™（作者开发）的敏捷开发工作流程，提供一个有明确作用流程框架的模式。不管怎样，设计模式超越了任何流程，无论怎样赞美它都不过分。

当阅读完本书后，你很可能成为职业的嵌入式系统开发者，并可用专业知识解决你在生活中遇到的实际设计问题。

作者简介

Bruce Powel Douglass 在俄勒冈大学获得运动生理学硕士学位，并在 USD 医学院获得神经生理学博士学位。在 USD 医学院时，他开创了一个叫做自相关因子分析的新数学分支，用于研究多细胞生物神经系统中的信息处理。

Bruce 拥有 30 余年的实时系统领域软件开发工作经验，并且是实时系统领域和系统工程领域知名的演说家、作家和咨询顾问。他是嵌入式系统大会顾问委员会的成员之一，并且教授软件估算和调度、项目管理、面向对象分析和设计、通信协议、有限状态机、设计模式、安全关键系统的设计等相关课程。他在实时面向对象分析和设计、项目管理领域有很多年的开发、讲课和咨询经验。他为许多刊物撰写文章，特别是在实时领域。

他是 IBM Rational 的首席技术宣传官，IBM Rational 是实时系统开发工具的主要生产商，产品包括广泛使用的建模工具 Rhapsody。Bruce 与其他 UML 伙伴合作共同制定了 UML 标准。他是对象管理组织的实时分析与设计工作组的前联合主席。他还著有一些与其他软件相关的书籍，包括：

- 《Doing Hard Time：Developing Real-Time Systems with UML，Objects，Frameworks，and Patterns》（Addison-Wesley，1999）
- 《Real-Time Design Patterns：Robust Scalable Architecture for Real-Time Systems》（Addison-Wesley，2002）
- 《Real-Time UML Third Edition：Advances in the UML for Real-Time Systems》（Addison-Wesley，2004）
- 《Real-Time UML Workshop for Embedded Systems》（Elsevier，2007）
- 《Real-Time Agility》（Addison-Wesley，2009）

目　录

第 1 章　什么是嵌入式编程

我们将学到：

- 嵌入式系统的基础知识
- 面向对象编程与结构化编程
- 使用 C 语言实现类、继承、状态机

1.1　嵌入式系统有何特殊之处

本书完全着眼于嵌入式系统的开发。为此，需要区分嵌入式系统和其他系统。在深入讨论之前，我们需要理解这种区别，这样才能领会用来开发嵌入式系统的那些模式和技术的真谛。

嵌入式系统可以定义为：不提供通用的计算环境，而是致力于完成现实世界中具体功能的计算系统。显然，这样定义的嵌入式系统非常宽泛，它包括了心脏起搏器中的微型 8 位嵌入式计算机，与控制航天设备相关的 32 位计算机，交通设施、飞行器的消防控制，以及 C^4ISR（Command，Control，Communications，Computers，Intelligence，Surveillance，and Reconnaissance，指挥、控制、通信、计算机、情报、监视和侦察）系统的广域网中用于战场管理的数以百计的强大计算机系统。很多嵌入式系统没有磁盘，没有人机交互，并且仅有很少的存储空间，但是嵌入式系统的市场空间却远比这些简单设备更宽广。

嵌入式系统无处不在：

- 在医药领域，嵌入式系统包括植入设备（如心脏起搏器、去纤颤器、胰岛素泵）、监测设备（如心电图（ECG/EKG）监视器、血气监测仪、血压计、肌电图（EMG）显示器）、成像系统（如 CT、SPECT、PET、TEM 和 X 射线成像仪），以及治疗输送装置（如病人呼吸机、药物汽化器和输液泵）。
- 在电信行业，涵盖了手机、交换设备、路由器、调制解调器和卫星等设备。
- 在汽车制造领域，嵌入式系统优化发动机燃烧，管理变速器中的能量传输，监控传感器数据，控制防抱死系统，提供安全保障并且提供咨询娱乐服务，例如：CD 和 DVD 播放器、GPS 导航（在一些地方，它可以提供雷达和激光探测甚至是主动雷达和激光探测应对系统）。
- 在办公领域，嵌入式系统管理电话、打印机、复印机、传真机、照明设备、数字投影仪、安保系统、防火系统和灭火系统。
- 在家中，例子包括微波炉、电视机、收音机、洗衣机，甚至包括真空吸尘器。

嵌入式系统已经能控制、增强、监控和管理几乎所有高科技设备，从电视机到火车，再到工厂自动化系统，而且其应用呈上升趋势。

嵌入式系统有个重要的子集即实时系统。很多人错误地认为“实时”就是“很快”，这种理解是不正确的。实时系统是一种在系统中必须满足实时性约束才能正确运行的系统。通常来看，可以将实时系统简单地分为两类：其中的一类是“硬”实时系统，它以系统实时性约束中的最后期限作为建模依据，在最后期限到来之前，指定的活动必须完成；“软”实时系统则是相对于

“硬”实时系统而言的[⊖]，它加入时间期限以外的标准（经常是随机的）来满足系统的实时性约束。这些标准包括平均吞吐量、平均执行时间、最大脉冲长度或其他的衡量标准。所有的系统都可以建模为硬实时系统，但是这样做经常造成“过度设计”，过度设计造成系统运行速度过快且拥有更多可用资源（实际上这些都是不需要的），同时也提高了系统的续生成本（recurring cost，近似于“制造成本”）。

尽管所有的系统都以硬实时系统作为建模模型，但事实上很多系统并不是真正的硬实时系统。如果系统响应偶尔延迟，甚至整个输入事件丢失，大多数系统将继续正常工作。把实时系统建模为“硬”实时的主要原因是，可以通过数学分析简化系统的实时性约束。

1.1.1 嵌入式设计约束

从系统内部看，嵌入式系统最突出的特点之一就是有严格的约束。与编写通用计算机软件不同，嵌入式系统发布时通常集成了系统所需的所有硬件。硬件平台通常不支持用户自行扩展，因此内存、电源、冷却器以及运算容量等资源构成了单位成本（续生成本）。为维持赢利，研发人员总是顶着极大的压力，尽量少使用这类硬件资源。这意味着嵌入式系统通常需要比桌面应用程序更多的额外的优化功能。

除了尽量减少使用硬件的需求外，性能问题也是一个系统成功的关键。性能包括很多方面，不同方面的性能在不同系统中的价值也各不相同。在某些系统中，吞吐量是一个至关重要的标准。吞吐量通常通过以下几个方面衡量：事务数、实例数、连接数和单位时间单元内的处理消息的数量。在其他一些系统中，快速响应每一个请求则更为重要。通常捕获系统最慢的响应时间作为系统的响应能力指标，其他系统值则通过预估超过最大吞吐量或系统的响应能力而获得。预估值可通过从正在发生事件的值域或概率密度函数中获得。

可靠性、健壮性、安全性是嵌入式系统应该遵守的其他约束。系统的可靠性用来（随机地）衡量系统正确完成功能的可能性。健壮性是指系统在违反运行的先决条件下（如操作条件、输入数据速率）仍能妥善提供相应的服务的能力。安全性表示系统的风险水平，也就是说使用该系统可能会造成哪些意外或损失。通常需要额外的硬件或者软件措施来解决这些问题，以维持系统操作运行在可接受的范围之内。例如，大多数嵌入式系统开机自检功能（power on self-test，POST）和定期或者连续的自检功能（built-in test，BIT）。

综合来说，这些系统中的约束就是所谓的系统提供的服务质量（QoS）。除了各种服务质量约束外，为减少续生成本，通常开发商会开发定制的硬件，并且为这些硬件提供专门的驱动软件支持。

1.1.2 嵌入式工具

很多嵌入式操作系统在目标机上执行，但是这些系统却是在其他系统（主机）上开发的。使用这种方法开发嵌入式系统对开发者和开发时使用的工具有很多影响。最明显的工具例子是交叉编译器。这种编译器在主机上运行，但是开发的可执行代码却在不同计算机和操作环境中运行。很多实时操作系统（Real-Time Operating System，RTOS）提供专有的编译器或者自定义的如 GCC（Gnu Compiler Collection，GNU 编译程序集[⊜]）这样的开源编译器。

⊖ 然而，俗话说：“实现硬实时系统很难，实现软实时系统更难”。

⊜ 参看 www.gnu.org 以获取最新版本和目标支持。

连接器是一个程序，它用来将一系列可执行代码合并连接到一个目标机可执行的目标文件中。一些操作系统不需要明确的连接步骤，因为当程序进入内存后，操作系统加载器会动态连接程序的相关元素。这是一种纯 UNIX 风格的嵌入式操作系统的做法，然而大多数嵌入式操作系统还是需要一个明确的连接步骤。连接器经常重定位程序，这意味着在连接步骤中和遇到汇编语言的跳转指令时，必须将原指定的起始地址映射到真实的起始地址上。

载入程序是一种工具，它将连接步骤中输出的对象映像载入到目标环境的内存中。这个工具有可能需要通过串口或者网络连接才能够将软件映像烧录到诸如 Flash 或 EPROM 这样非易失性存储器上。作为将软件映像加载到目标平台的另一种选择，很多开发者使用模拟器在主机开发系统上运行目标程序。例如，通常在 Windows 中使用 Z80 或 8051 模拟器运行、调试、测试软件，这些过程甚至发生在目标板可用之前。

到目前为止，工具链中提到的工具仅仅能将软件加载到系统中。除此之外，我们需要确保软件正常工作。接下来介绍的工具集是调试器工具集，通过使用这些工具，我们能很好地对可执行的软件进行测试，包括单步执行（一行一行执行）或者逐过程执行（全部执行），设置断点，以及查看和修改变量。这些调试器可以通过标准串口和网络连接或 JTAG[⊖]测试端口连接执行。

现代集成开发环境（Integrated Development Environment，IDE）将绝大多数嵌入式开发工具链中的工具集成到一个环境中，从而使开发过程变得快捷和自动化。最新的工业发展趋势是将集成开发环境转移到 Eclipse[⊜]平台下，因为 Eclipse 有强大的集成环境以及可用的第三方或开源组件。Jazz 基金会将利用 Eclipse 平台进一步集成管理、测试、报告和其他工具，以便于更好地支持协同软件的开发和交付。

1.1.3 OS，RTOS，还是没有操作系统

操作系统（Operating System，OS）为应用程序开发人员提供一组系统和平台执行服务，特别是在目标系统资源的管理和使用方面。这些资源包括内存、并发程序（进程、任务或线程）、事件队列、中断、硬件和应用程序。大多数操作系统不能保证实时性，而且桌面操作系统可能由于内部处理、内存管理或未知次数的垃圾回收触发不可预见的延迟。这种不可预测性和桌面系统十分庞大（与有限内存的系统相比）的事实，使得它们不适合在时间和资源都受制约的实时嵌入式环境下运行。

实时操作系统（RTOS）是一个多任务的操作系统，运行实时和嵌入应用程序。实时操作系统提供高效、高性能的服务，但是通常性能的预测比最大吞吐量更重要。此外，与桌面操作系统相比，实时系统更轻巧但功能稍差。实时操作系统不能保证提供实时性能，但是它可以提供一个应用程序的运行环境，使得正确开发的应用能实现实时的性能。

实时操作系统使用三种基本的设计模式中的一种来实现程序和任务的运行。当事件出现时，事件驱动的系统通过任务调度处理出现的事件。当多个任务准备运行时，大多数事件驱动的系统依据任务优先级来决定运行哪个任务。任务优先级通常是静态的（例如，在设计阶段指定为单调速率调度（rate-monotonic scheduling），但是有一些优先级则是动态分配的，系统根据当前的操作

⊖ 全称 Joint Test Architecture Group，它是指很多开发人员用于在嵌入式目标系统中执行测试的电子和软件的标准接口。参看 www. freelabs. com/%26whitis/electronics/jtag/或 www. embedded. com/story/OEG20021028S0049 获取更多信息。

⊜ 网址 www. eclipse. org。

情况，动态改变任务的优先级（例如，最早时限优先调度（earliest deadline first scheduling）。其他两个任务调度的方法使用“公平主义原则”，要么给所有的任务一个时间片来运行（基于时间基础，如轮询调度（round robin scheduling），要么按周期调度任务（基于序列，如循环调度（cyclic executive scheduling））。

20年前，多数开发公司专注于开发自己的RTOS，但是RTOS的市场已经经历了整合，致使这些公司转而购买商业RTOS，而花更多时间努力地开发自己的专业产品。

即使是一些大企业，如Wind River（VxWorks的开发商）和Green Hills软件公司（Integrity的开发商），在使用实时操作系统时也面临一些挑战。目前使用的实时操作系统有上百种之多，而且每个操作系统都有自己的应用编程接口（Application Program Interface，API）。此外，虽然实时操作系统在发展和实时性上没有桌面操作系统影响那么深远，但这并不代表没有影响。很多嵌入式操作系统还在8位的处理器和仅有数千字节的内存（对于标准的实时操作系统来说内存太小）的设备上运行。很多实时操作系统支持一系列可扩展的服务（基于微内核架构模式[㊀]），但是可扩展性的精确程度可能不能满足严格限制的应用程序。

有些系统由于资源太少而不能支持RTOS和实际的应用程序，转而选择不用操作系统，也就是说不使用商业或专有的RTOS而完成相应的功能。这意味着应用程序代码本身必须复制一些RTOS的服务或应用程序来运行所需的功能。随着硬件成本的降低，系统由原来仅8位的处理器转向更大的16位或32位的处理器。这就需要添加现代嵌入式设备所需的额外复杂的行为，而且还需要更低的制造和部件的续生成本。

对于非常简单的嵌入式系统，可能没有明确的操作系统功能。应用程序可能仅仅是由通过分享资源模式如队列或内存共享通信的一组中断处理。另外，一个简单的循环执行的任务循环实现可能就足够了。稍微复杂一些的系统可能有如内存或任务管理这样的操作系统功能，因为它们需要这些功能。

从某个角度来看，操作系统是一整套提供一定的服务和资源给应用程序的设计模式。本书第3章讨论访问硬件资源，如定时器和内存的模式，而第4章关注管理嵌入式系统中的并发方法。

1.1.4 嵌入式中间件

中间件是一种使用某种方法将软件组件连接的软件。中间件作为一个术语可追溯到1968年NATO软件工程会议[㊁]。像操作系统一样，中间件有商业支持，但是在小型系统中，它可能作为应用软件的一个部分而被开发。最普通的实时和嵌入式系统的中间件包括通用对象请求代理体系结构（Common Object Request Broker Architecture，CORBA）和它的衍生结构，还有数据分发服务（Data Distribution Service，DDS）。这些中间件架构是基于对象管理组织（Object Management Group，OMG[㊂]）公布的标准。在CORBA这个架构中，有很多专有的衍生标准可供选择，包括实时CORBA、嵌入CORBA和最小化CORBA。

你可能会猜，中间件适合更大规模的由多个软件组件和应用组成，分布在多个处理器和网络上的嵌入式应用。尤其当组件由不同的组织开发，系统将被不同的组织扩展，或者当系统有很长

㊀ 参见我的书《Real-Time Design Patterns：Robust Scalable Architecture for Real-Time Systems》（Addison-Wesley，2002）。

㊁ http://homepages.cs.ncl.ac.uk/brian.randell/NATO/nato1968.PDF

㊂ 网址 www.omg.org。

的生存期时，使用标准中间件可以给开发者提供显而易见的好处。

这些中间件架构，又像是操作系统，是设计模式的完整集合，例如代理模式（Proxy）、数据总线模式（Data Bus）和中介模式（Broker Pattern）。

1.1.5 与硬件协同开发

很多嵌入式工程涉及电子和机械硬件与软件开发同时进行。这给软件开发者带来了特殊的挑战，开发者仅根据硬件如何工作初步设计规格说明来开发软件。通常，这些规格说明仅仅就是功能的概念说明，这使得，不到硬件最终发布，就不可能开发出正确的软件。任何硬件计划安排的失误会直接导致软件计划安排的失误，为此软件开发者必须顶得住压力。

这使我想起了20世纪80年代为遥感系统制定硬件规格的工作。在那个系统中硬件应该是位脉冲编码，8个脉冲代表“0”位，15个脉冲代表“1”位，脉冲之间有具体时间，并且位之间的时间更长。我按照这个规格说明开发软件，但是令我惊讶[㊀]的是，当拿到实际硬件时，软件仅能断断续续地与设备进行通信。尽管进行了深入细致的测试，我还是不能指出软件问题到底出在什么地方，直到我拿到示波器，才解决这个问题。通过示波器我发现当软件输出“0”位时，线圈各处发射脉冲6～12次。对于“1”位，它各处发射脉冲10～18次（注意“0”和“1”的范围实际上重叠了）。解决方案是写一个预测校正算法，它基于消息的CRC估计可能的位误差并在传输时的纠正位值。软件确实比我想象的更加复杂！正如嵌入式程序员的生活一样。

虽然很多软硬件集成的痛苦可以通过优秀的规格说明和硬件的早期原型发布得以消除，但是很多痛苦还是不能消除。软件开发的现实之一是，协同开发的结果是需要再工程。

敏捷方法通过几种不同的方法[㊁]解决这个问题。首先，通过使用敏捷方法，软件在整个软件生命周期中持续开发。每天都编写、调试、单元测试、发布软件，并且每天集成不同的软件单元，并在工作流中作为一个紧密结合的整体进行测试，也称为持续集成（Continuous integration）。这样能及早发现软件缺陷。这种集成也能（或应该）包括早期的硬件。

1.1.6 调试和测试

普鲁士将军von Clausewitz[㊂]曾经写道：“战争中的一切都非常简单，但是最简单的事情却非常难。”关于软件他很可能也会说这样的话！

关于软件的困难的简单事不是编写软件，而是编写有正确功能的软件。开发无缺陷软件最先进的方法是称为测试驱动开发（Test-Driven Development，TDD）的敏捷开发。在TDD中，编写将验证软件部分的单元测试与软件开发同时进行，甚至稍微提前。普遍情况下，单元测试完全被忽略或者执行太晚以致会对软件产生不利的影响。

通常有很多种适用于软件的单元测试，特别是嵌入式软件。它们包括：

- 功能——测试系统或系统单元的行为或功能。
- 服务质量——测试系统或系统单元的“性能”，通常通过系统性能需求测量系统或单元的性能。

㊀ 那时我太天真。;－）

㊁ 参看我的书《Real-Time Agility》（Addison-Wesley，2009）中针对这个话题的详细讨论。

㊂ von Clausewitz，C.，1976。《战争论》，Howard，M.，Paret，P.（编辑翻译），普林斯顿大学出版社，1832第1版，原名为《Vom Kriege》。

- 先决条件测试——在系统预设约束条件得到满足或预设约束条件被破坏时，测试系统和系统单元的正确性。
- 域——测试数据域中的值。
- 统计——通过从概率密度函数（Probability Density Function，PDF）中随机选择一些值，测试这些值是否在一个值域内。
- 边界——测试在值域的边界、内部、外部的值。
- 覆盖——在测试集中测试所有可执行的路径。
- 压力——测试数据超出了系统或系统单元期望的带宽。
- 容积——也称作“负载测试”，使用达到或超过设计负载的巨大数据测试系统。
- 错误播种——有意将故障引入到系统中的测试，来确保系统能正确处理。
- 回归测试——通常是以前通过测试的一个子集，确保修改后的系统没有引入新的错误到之前有正确功能的系统中。

在桌面系统中，测试用例完全涵盖这些关注点的单元测试软件非常罕见（且非常困难）。当你在弱访问目标平台上嵌入这种软件就变得更困难。可以采用一些不同的策略执行测试用例，例如：

- “printf”测试——通过写入一个文件或标准输出来测试系统。
- “伙伴测试”——编写的测试附加模块将测试用例嵌入到它们的功能中。
- 主机上测试——在主机平台上使用主机原生编译器执行大多数测试并在目标平台上使用交叉编译器执行一个关键的子集。
- 主机模拟——通过使用交叉编译软件，在主机上模拟目标平台并且用相同的对象代码在目标机上重测关键部分。
- 商业软件测试工具——使用软件测试工具，如 TestRT、LDRA 或 VectorCAST。
- 商业软硬件集成工具——这些工具包括如逻辑分析器、线路内仿真器、JTAG 兼容测试工具和 ROM 模拟器。

在多种多样的测试中，性能测试通常是最难充分执行的。因为执行测试用例大多数通用的做法涉及在同一个目标平台上运行额外的软件，这会在测试中影响软件的时序和性能。如果可能的话，最好使用线路内仿真器辅助硬件调试或为性能测试使用 ROM 模拟器。

1.2 面向对象还是结构化

结构化编程是一种强调分开和有区别的两方面都有规可循的软件开发形式。

一方面，函数和过程形成基本的编程基础：过程是具有单一入口的简单行为的集合，执行一致的行为目标；函数是有返回值的简单过程。过程可以分解到调用树中，通过一个过程调用另一个过程，允许分解算法。过程通常同步调用（呼叫），但是通过添加额外的手段，也可以执行异步调用。

结构化编程的另一方面是数据结构的概念。所有第三代计算机语言都有从更基本类型元素创建复杂数据结构的概念，最终建立在计算机语言提供的基本类型上。这些数据结构可能是同构集合，如数组，也可能是异构的，像是 C 语言的结构体。

面向对象编程基于正交范式（orthogonal paradigm）。面向对象编程仅有基于类概念的一个分类标准，而不是两个单独的分类标准。类将数据（存储为属性）和在数据上执行操作的过程（称为操作）组合到一起，因为它们往往是内在紧密结合的。对象是类的实例。这使对象相当于结构化

语言中的变量，但是它功能更强大，因为对象提供属性的值和操纵它们的操作。

结构化语言是直接支持结构化编程的语言。在本书涉及的内容中，C 语言明显是结构化的语言。C 是到目前为止创建嵌入式系统使用最普遍的语言，而且 C 是固有的“结构化”语言。它具有所有特征，如函数、变量等。然而，出现了一个有趣的问题：像 C 一样的结构化语言能够完成面向对象编程吗？而且即使可以做到，是否应该做？

我回想起了类似的讨论，当时最常见的编程语言是汇编语言。用汇编代码可以编写结构化的程序吗？毕竟，汇编语言没有提供结构化本质的特征。当它发生时，答案很清楚，是可以的！它不仅可以，而且可以很不错地编写结构化的汇编语言程序。这需要程序员遵守一些规则，并且也需要有将汇编语言指令映射到结构化概念（如在栈中传参数）的决策，但这是很容易做到的。

相同点确实是面向对象编程蕴含在像 C 一样的结构化语言中。在本书中，我们将在编程中主要从基于对象的角度（不子类化的面向对象）讨论，因为我相信这样做是有好处的。但是，使用 C 创建基于对象或面向对象的程序是非常明确的。让我们简短地讨论一下面向对象编程的重要方面，以及如何使用 C 来实现这些概念。

1.2.1 类

类仅是一个 C 语言结构体，但特殊之处是包含两种不同的特性：数据（属性）和行为（操作）。

最简单实现类的方法是简单使用文件作为封装边界；公共变量和方法在头文件中可见，而在实现文件中包含方法体、私有变量和方法。多个文件可使用《Usage》依赖关系连接以支持调用树，如图 1-1 所示。在这个例子中，“类”表现为一对文件并且它的实现使用一些 Display 类（文件）的特性（变量和方法），在这个例子中是 `displayMsg()`方法。

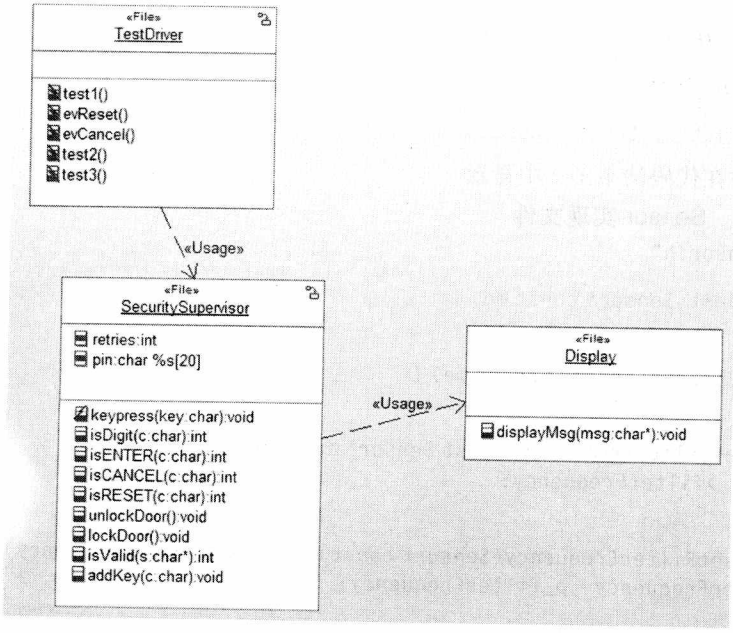

图 1-1 UML 中用文件表现类

一个更为灵活的方式是使用文件内的结构体来表示类。类的操作用位于相同文件内的结构体的函数定义。为确保函数能访问正确的对象数据，我们需要传递一个 me 指针。

这允许我们有同一个类的多个实例（对象），并且保证成员函数在正确的数据拷贝上工作。此外，类可以赋予“特殊”的操作。构造函数创建类的一个对象。初始化程序（可选择的）初始化对象和它的属性。析构函数销毁类并释放已使用的内存。

考虑一个简单类 Sensor，它有数据元素 value、updateFrequency、filterFrequency，操作 getValue()、setValue(v:int)、setUpdateFreq(r:int)、getUpdateFreq()、setFilterFreq(ff:int)和 getFilterFreq()。头文件如代码清单 1-1 所示。

代码清单 1-1 Sensor 头文件

```
#ifndef Sensor_H
#define Sensor_H

/*## class Sensor */
typedef struct Sensor Sensor;
struct Sensor {
    int filterFrequency;
    int updateFrequency;
    int value;

};

int Sensor_getFilterFrequency(const Sensor* const me);

void Sensor_setFilterFrequency(Sensor* const me, int p_filterFrequency);

int Sensor_getUpdateFrequency(const Sensor* const me);

void Sensor_setUpdateFrequency(Sensor* const me, int p_updateFrequency);

int Sensor_getValue(const Sensor* const me);

Sensor * Sensor_Create(void);

void Sensor_Destroy(Sensor* const me);
```

相关实现文件在代码清单 1-2 中显示。

代码清单 1-2 Sensor 实现文件

```
#include "Sensor.h"

void Sensor_Init(Sensor* const me) {
}

void Sensor_Cleanup(Sensor* const me) {
}

int Sensor_getFilterFrequency(const Sensor* const me) {
    return me->filterFrequency;
}

void Sensor_setFilterFrequency(Sensor* const me, int p_filterFrequency) {
    me->filterFrequency = p_filterFrequency;
}
int Sensor_getUpdateFrequency(const Sensor* const me) {
```

```
    return me->updateFrequency;
}

void Sensor_setUpdateFrequency(Sensor* const me, int p_updateFrequency) {
    me->updateFrequency = p_updateFrequency;
}

int Sensor_getValue(const Sensor* const me) {
    return me->value;
}

Sensor * Sensor_Create(void) {
    Sensor* me = (Sensor *) malloc(sizeof(Sensor));
    if(me!=NULL)
        {
            Sensor_Init(me);
        }
    return me;
}

void Sensor_Destroy(Sensor* const me) {
    if(me!=NULL)
        {
            Sensor_Cleanup(me);
        }
    free(me);
}
```

在这个实例中，文件充当封装边界，且分开两种不同的文件类型（头文件即 .h 文件和实现文件即 .c 文件）将元素封装在单个类中。这种方式较好地支持了基于对象编程，但是不能以在子类中简单重载的方式实现虚拟函数。

另一种可选择的支持面向对象编程的方法是在结构体本身嵌入函数指针。这种方法将在 1.2.3 节讨论。

1.2.2 对象

对象是类的实例，同样的特定变量 x（int 类型）是它类型（int）的一个实例。相同的操作符集合（如算术运算）适用于所有这种变量类型，但是一个实例的特定值与另一个实例是不同的（叫做 y）。

在标准 C 编程中，复杂算法仍能嵌入到类中，但是通常这些类是单例的（singletons），这意味着在应用程序中，类仅有一个实例。例如单例类 Printer 可能有如 currentPrinter 的变量和如 print() 的操作，但是应用程序将只有一个单一实例。即使仅有一个实例运行，但是通过类封装数据和对这些数据的操作仍有好处。在其他情况下，通常更多的是以数据为中心（而不是以算法或服务为中心）的类将有很多实例。

创建类的实例，仅是创建了结构体的一个实例。考虑一个可能的 main() 函数，它创建和使用两个前面 Sensor 类的两个实例，如代码清单 1-3 所示。

代码清单 1-3 Sensor main()

```
#include "Sensor.h"
#include <stdlib.h>
```