第11章 软件实现

本章内容：

第11章 软件实现

软件实现就是在下列细节设计的基础上，用一种程序设计语言来将设计转换为程序，得到的结果是源程序代码。因此在软件实现前，一定要熟练程序设计，了解程序设计语言的特色及编码，以及应注意程序设计风格。

11.1 程序设计语言的特性及选择

程序设计语言是人机通信的工具之一。每种程序设计语言都有各自的特点，这些特点也会对人思考和解决问题产生影响。

11.1.1 程序设计语言特性

在程序设计前充分了解语言的特点，对成功和高效开发软件有重大的影响。下面从三个方面介绍语言的特性。

1. 工程特性

从软件工程的观点、程序设计语言的特性着重考虑软件开发项目的需要，因此对程序编码有如下要求：

1）可移植性

指程序从一个计算机环境移植到另一个计算机环境的难易程度，计算机环境是指不同硬件、不同的操作系统版本。一般来说，程序设计中要尽量避免直接对硬件进行操作，要使用标准的程序设计语言和标准的数据库操作，尽量不使用扩充结构。对程序中各种和硬件、操作系统有关的信息，使用参数化的方法，提高通用性。

2）开发工具的可利用性

指软件开发工具在缩短编码时间，改进源代码的质量方面的能力。目前，许多编程语言都有一套集成的软件开发环境。这些开发工具为源程序代码编写提供各种库函数、交互式调试器、报表格式定义工具、图形开发环境、菜单系统和宏处理程序等。

3）软件的可重用性

指编程语言提供可重复使用的软件成分的能力，如模块化的程序可通过源代码剪贴、使用继承方式实现软件重用。提供软件重用性的程序设计语言可以大大提高源程序的利用率。

4）可维护性

指将详细设计转变为源程序的能力和对源程序进行维护的方便性。因此，将设计文档转化为源程序特性、源程序的可读性和语言的文档化特性对软件的可维护性具有重大的影响。

2. 技术特性

在确定了软件需求之后，所选择的语言的技术特性会对软件工程的其余阶段有一定的影响，因此要根据项目的特点选择相应的语言，有的要求实时处理能力强，有的要求对数据库进行很方便地操作，有的要求能对硬件做一些操作。一般在软件设计阶段的设计质量与语言的关系不大（面向对象设计除外），但在编码阶段，质量往往受语言特点的影响，甚至可能会影响到设计阶段的质量。

如面向对象的语言可以提供抽象类、继承等方法， 而JAVA会提供关于网络设计方面的很多工具，而汇编语言可以直接对机器硬件进行操作。当选择了一种语言后，就可以影响到对概要设计和详细设计的实现。语言的特性对于软件的测试与维护也有一定的影响，支持结构化构造的语言有利于减少程序环路的复杂性，使程序易测试、易维护。

3. 心理特性

语言的心理特性指影响程序员心理的语言性能。程序的实现最终要靠人来实现，因此人的因素对程序的实现质量是有很大的影响的。而程序语言的心理特点，主要是表现在编码实现时对人的影响，包括对程序代码的理解等。它在语言中的表现有以下几个方面。

1）歧义性

对于一个程序代码，不同的人对它的意义有不同的解释，这就是心理上的二义性。但是实际上编译程序总是根据语法，按一种固定方法来解释语句，不存在二义性。

如：X=X1/X2·X3，编译系统只有一种解释，但人们却有不同的理解，有人理解为X=(X1/X2)·X3，而另一个人可能理解为X=X1/(X2·X3)。又如FORTRAN语言中变量的类型有显式定义和隐式定义两种，用REAL K显式说明K是实型变量，但按隐含类型定义，K是整型变量。当程序很长时，将会使程序员不了解它的数据类型而产生错误。这样的程序可读性也较差。

2）简洁性

人们要掌握一种语言，就要记住语句的种类、各种数据类型、各种运算符、各种内部函数和内部过程，这些成分数量越多，简洁性越差，人们越难以掌握。但过分简洁会造成程序难以理解，一致性差。因此语言的简洁必须有一个合适的度。

3）局部性和顺序性

程序语言的局部性是人的记忆的联想方式的表现。在编码过程中，由语句组合成模块，由模块组装成系统结构，并在组装过程中实现模块的高内聚，低耦合，使局部性得到加强。顺序性是人的记忆的顺序方式的表现。人的顺序记忆提供了回忆序列中下一个元素的手段，对于具有一定顺序规律的事物，人是容易记忆的。人的记忆特性对使用语言的方式有很大的影响。

4）传统性

指人们在学习新的内容时比较容易受到已经有的内容的影响。而传统性的表现影响人们学习新语种的积极性，若新语种的结构、形式与原来的类似，比较容易接受，若风格和设计思想差别很大，则在学习新的语言时，原有的语言知识会起到阻碍的作用。比如学习BASIC语言的人，来学习C++语言，势必会遇到很大的困难。

11.1.2 程序设计语言的选择

为开发一个特定项目选择程序设计语言时，必须从技术特性、工程特性和心理特性几方面考虑。在选择语言时，要从问题需求入手，确定它的要求是什么，以及这些要求的相对重要性，针对这种需求，需要什么特性的程序设计语言来实现。由于一种语言不可能同时满足它的各种需求，所以要对各种要求进行权衡，比较各种可用语言的适用程度，最后选择认为是最适用的语言。选择语言时，可以从以下几个方面来考虑：

1. 项目的应用领域

项目应用领域是选择语言的关键因素，有下列各种类型。

1）科学工程计算

该计算需要大量的标准库函数，以便处理复杂的数值计算，可供选用的语言有：

（1）FORTRAN语言：是世界上第一个被正式推广应用的计算机语言，产生于1954年，经过FORTRAN 0到FORTRAN 4，又相继扩展为FORTRAN 77，FORTRAN 90，通过几个版本不断的更新，使它不仅面向科学计算，数据处理能力也极强。

（2）Pascal语言：产生于19世纪60年代末，具有很强的数据和过程结构化的能力，它是第一个体现结构化编程思想的语言，由于它语言简明、数据类型丰富、程序结构严谨，许多算法都用类Pascal来概括。用Pascal语言写程序，也有助于培养良好的编程风格。

（3）C语言：产生于20世纪70年代初，最初用于描述UNIX操作系统及其上层软件，后来发展成具有很强功能的语言，支持复杂的数据结构，可大量运用指针，具有丰富灵活的操作运算符及数据处理操作符。还可以直接对位进行操作，程序运行效率高。

（4）PL/1语言：是一个适用性非常广泛的语言，能够适用于多种不同的应用领域，但由于太庞大，难以推广使用，目前一些PL/1的子集被广泛使用。

（5）C++语言：支持面向对象的设计思想，支持继承和多态性等概念，可以大大提高程序的重用性，是现代软件设计的趋势。

2）数据处理与数据库应用

数据处理与数据应用可供选用的语言如下：

（1）COBOL语言：产生于20世纪50年代末，是广泛用于商业数据处理的语言，它具有极强的数据定义能力，程序说明与硬件环境说明分开，数据描述与算法描述分开，结构严谨，层次分明，说明采用类英语的语法结构，可读性强。

（2）SQL语言：最初是为IBM公司开发的数据库查询语言，目前不同的软件开发公司有不同的扩充版本，如20世纪80年代后期我国引入Informix_SQL，Microsoft的SQL可以方便地对数据库进行存取管理。

（3）4GL语言：称为第4代语言，随着信息系统的飞速发展，原来的第2代语言（如FORTRAN、COBOL）第3代语言（如Pascal、C等）受硬件和操作系统的局限，其开发工具不能满足新技术发展的需求，因此，在20世纪70年代末，提出了第4代语言的概念，4GL的主要特征是：

① 友好的用户界面：指操作简单，使非计算机专业人员也能方便地使用它。

② 兼有过程性和非过程性双重特性：非过程性指语言的抽象层次又提高到一个新的高度，只需告诉计算机“做什么”，而不必描述“怎么做”，“怎么做”的工作由语言系统运用它专门领域的知识来填充过程细节。

③ 高效的程序代码：指能缩短开发周期，并减少维护的代价。

④ 完备的数据库：指在4GL中实现数据库功能，不再把DBMS（数据库管理系统）看成是语言以外的成分。

⑤ 应用程序生成器：能提供一些常用的程序来完成文件维护、屏幕管理、报表生成和查询等任务，从而有效提高软件生产率。

目前流行的Microsoft公司的FoxPro、Uniface公司的Uniface、Powersoft公司的Power Builder、Informix公司的Informix_4GL以及各种扩充版本的SQL等都不同程度地具有上述特征。

3）实时处理

实时处理软件一般对实时性能的要求很高，可选用的语言有：

（1）汇编语言：是面向机器的，它可以完成高级语言无法满足的特殊功能，如与外部设备之间的一些接口操作。

（2）Ada语言：是美国国防部出资，在Pascal语言基础上开发出来的，主要用于适时、并发和嵌入系统的语言。它特别地支持程序的实时设计要求，包括多任务处理、中断处理、任务间同步与通信等。到现在已经发展成为安全、高效和灵活的面向对象的编程语言。

4）系统软件

系统类软件很多时候都需要同计算机的硬件打交道，因此在编写操作系统、编译系统等系统软件时，可选用汇编语言、C语言、Pascal语言和Ada语言。

5）人工智能

如果要完成专家系统、推理工程、语言识别、模式识别、知识库系统、机器人视角及自然语言处理等和人工智能有关的系统，可以选择以下的几种语言：

（1）**LISP：**是一种函数型语言，产生于20世纪60年代初，它特别适用于组合问题中的符号运算和表处理，广泛应用于推理证明、树的搜索等问题。近年来ISP广泛应用于专家系统的开发，对于知识库系统中的事实、规则和推理有很好的支持。

（2）**PROLOG：**是一种逻辑型语言，产生于20世纪70年代初，可以较好地表示事实、规则和推理。这些子句组成一个个的程序，程序的风格接近于自然语言，比较符合人的思维方式。

以上讨论了一些典型的语言的特点和它们的适用领域，但在具体的软件设计情况下，很多时候并不能简单的就确定采用哪种语言，要根据具体情况，权衡各种要求，选择一种合适的程序设计语言。

2. 软件开发的方法

有时编程语言的选择依赖于开发的方法，采用4GL语言适合用快速原型模型来开发。如果是面向对象方法，就必须采用面向对象的语言编程。近年来，推出了许多面向对象的语言，这里主要介绍以下几种：

（1）C++：是由美国AT&T公司的Bell实验室最先设计和实现的语言，它支持并实现了面向对象设计中的类的定义、继承、封装概念，并且与C语言兼容，大量的已经开发的C库、C工具以及C源程序不用修改就可以在C++里运行。因此，编程人员不必放弃自己熟悉的C语言，只需补充学习C++提供的那些面向对象的概念和程序设计方法就可以从C过渡到C++，加之它的开发效率和程序运行的效率较高，成为当今最受欢迎的面向对象语言之一。

（2）**Java：**是由Sun公司开发的一种面向对象的、分布式的、安全的程序设计语言。因为它运行在Java虚拟机上，因此它是与硬件无关的，也体现了它的易移植性。和C++比较，它不支持运算符重载、多继承等特性，但增加了内存空间自动垃圾收集的功能，使程序员不必考虑内存管理问题。Java应用程序提供了许多适合网络编程的对象。

3. 软件开发的环境

良好的编程环境不但有效提高软件生产率，同时能减少错误，有效提高软件质量。近几年推出了许多可视化的软件集成开发环境，特别是Microsoft公司的Visual BASIC、Visual C、Visual FoxPro及Borland公司的Delphi（面向对象的Pascal）等，都提供了强有力的调试工具，帮助用户快速形成高质量的软件。

4. 算法和数据结构的复杂性

科学计算、实时处理和人工智能领域中的问题算法较复杂，而数据处理、数据库应用和系统软件领域内的问题，数据结构化比较复杂，因此选择语言时可考虑是否有完成复杂算法的能力，或者有构造复杂数据结构的能力。

5. 软件开发人员的知识

软件开发人员原有的知识和经验对选择编程语言也有很大的影响。一般情况下，软件编程人员愿意选择曾经成功开发过项目的语言。新的语言虽然有吸引力，也会提供较多的功能和质量控制方法，但软件开发人员若熟悉某种语言，而且有类似项目的开发经验，往往愿选择原有的语言。为了能选择更好的适应项目的程序设计语言，开发人员应该经常学习新的程序设计语言，掌握新技术。

11.2 程序设计风格

开发软件项目过程中，测试和维护都是很重要的一个阶段。不论测试与维护，都必须要阅读源程序。因此，阅读程序是软件开发和维护过程中的一个重要组成部分。因为对源程序中的变量和语句所表达的实际意义不了解，对于技巧性强的程序，读程序的时间比写程序的时间还要多，并且很难理解。

一个程序的主要目的就是给其他人阅读，可以通过养成良好的程序书写风格来解决阅读性差的问题。程序设计风格指一个人编制程序时所表现出来的特点、习惯及逻辑思路等。一个公认的、良好的编程风格可以减少编码的错误，减少读程序的时间，从而提高软件的开发效率。良好的编码风格有以下几个方面：

1. 源程序文档化

编写源程序文档化的原则为：

1）标识符应尽量具有实际意义

若是几个单词组成的标识符，每个单词第一个字母大写，或者之间用下划线分开，这样便于理解。如某个标识符取名为rowofscreen，就不容易理解，但写成RowOf_Screen或row_of_screen就容易理解了。但标识符太长，虽然容易体现实际意义，但书写与输入都易出错，因此要在标识符长度和实际意义之间有一个好的平衡。

2）程序应加注释

阅读并能理解一个没有注释的程序是不可想象的。注释是程序员之间通信的重要工具，一般用自然语言或伪代码描述。注释的作用在于它说明了程序的功能，变量的实际意义。特别是对于维护阶段，注释是理解程序的重要途径。注释可分为序言性注释和功能性注释。序言性注释应置于每个程序或子程序的起始部分，它的主要内容有：

（1）说明每个模块的用途、功能。

（2）说明模块的接口形式、参数描述及从属模块的清单。

（3）该模块的数据描述：特殊的数组或变量的说明、约束或其他信息。

（4）开发历史：指程序的编写者、审阅者姓名及日期、修改说明及日期。

功能性注释嵌入在源程序内部，说明程序段或语句的功能以及数据的状态。加入功能性注释的原则有以下几点：

（1）只给重要的、理解困难的程序段添加注释，而不是每一行程序都要加注释。

（2）书写上要注意形式，以便很容易区分注释和程序。

（3）修改程序时，要注意修改相应的注释部分。

2. 数据说明

为了使数据定义更易于理解和维护，一般有以下的书写原则：

（1）数据说明顺序应规范，将同一类型的数据书写在同一段落中，从而有利于测试、纠错与维护。例如按常量说明、类型说明、全程量说明及局部量说明顺序。

（2）当一个语句中有多个变量声明时，将各变量名按字典顺序排列，便于查找。

（3）对于复杂的和有特殊用途的数据结构，要加注释，说明在程序中的作用和实现时的特点。

3. 语句构造

语句构造的原则为：简单直接，使用规范的语言，在书写上要减少歧义。不要一行多个语句，造成阅读的困难。不同层次的语句采用缩进形式，使程序的逻辑结构和功能特征更加清晰。要避免复杂、嵌套的判定条件，避免多重的循环嵌套，一般嵌套的深度不要超过三层。表达式中多使用括号以提高运算次序的清晰度，不要简单地依靠程序设计语言自身的运算符优先级等。

4. 输入和输出

在编写输入和输出程序时考虑以下原则：

（1）输入操作步骤和输入格式尽量简单，提示信息要明确，易于理解。

（2）输入一批数据时，尽量少用计数器来控制数据的输入进度，使用文件结束标志。

（3）应对输入数据的合法性、有效性进行检查，报告必要的输入信息及错误信息。

（4）交互式输入时，提供明确可用的输入信息。

（5）当程序设计语言有严格的格式要求时，应保持输入格式的一致性。

输入、输出风格还受其他因素的影响，如输入、输出设备，用户经验及通信环境等。

5. 效率

效率一般指对处理机时间和存储空间的使用效率，对效率追求要注意下面几个方面：

（1）效率是一个性能要求，需求分析阶段就要对效率目标有一个明确的要求。

（2）追求效率应该建立在不损害程序可读性或可靠性基础之上。在程序可靠和正确的基础上追求效率。

（3）选择良好的设计方法才是提高程序效率的根本途径，设计良好的数据结构与算法，都是提高程序效率的重要方法。编程时对程序语句做调整是不能从根本上提高程序效率的。

总之，在编码阶段，要善于积累编程经验，培养和学习良好的编程风格，使程序清晰易懂，易于测试与维护，从而提高软件的质量。

11.3 程序设计效率

一个良好的工程系统的标准是要求各种资源的使用达到临界状态。而处理器的时间周期和内存的利用通常被看做是临界资源，而编码则被看做是能节省出几微秒或几位的最后的地方。尽管效率（efficiency）是值得追求的目标，但是不能一味的追求效率而损害了程序的质量。

首先，效率是一种性能需求。因此在软件需求分析阶段就应该根据实际情况来规定。软件效率只要满足实际的要求就可以了，而不是越高越好。其次，一个良好的程序设计是提高效率的根本途径。最后，要知道代码效率与代码的简单性紧密联系。

总之，不要为了追求非必需的效率提高而牺牲代码的清晰性、可读性和正确性。

11.3.1 代码效率

对代码效率影响最大的是详细设计阶段所确定的算法的效率。除此之外，编码风格也会影响运行速度和对内存的需要，它的影响体现在以下的几个方面：

（1）在进行编码以前，应简化算法中的算术表达式和逻辑表达式，使之显得简洁。

（2）对嵌套循环仔细审查，在循环内部的语句和表达式越少越好。

（3）应尽量避免使用多维数组。

（4）应尽量避免使用指针和复杂的列表。

（5）采用效率高的算术运算。

（6）要避免采用混合数据类型。

（7）只要有可能，就应当采用占用内存少的数据类型。

如果是使用折叠的重复表达式、循环求值、快速算术运算，以及其他一些高效的算法，许多编译程序都可以自动地生成高效的目标码。对于那些要求特别高效的应用，这种编译程序是不可缺少的编码工具。

11.3.2 内存效率

由于硬件技术的发展，内存的容量增加得很快，因此在现在的微机中，大多数程序都不用考虑内存限制了。而虚拟内存管理技术又为应用软件提供了巨大的逻辑地址空间，在这种情况下，内存效率不等于使用节约内存。内存效率必须注意考虑操作系统内存管理的分页特征，而根据代码的局域性或通过结构化构造功能域的设计方法才是减少程序在运行时产生频繁的页面置换和提高内存效率的最好办法。

虽然低成本、大容量的内存芯片发展很快，但在嵌入式微处理器领域中，内存限制仍是一个非常实际和不得不考虑的问题。如果系统需求要求最小内存，那就得必须非常细心地对高级语言编译后的目标代码进行估算，甚至采用汇编语言。在嵌入式中，提高运行效率的技术往往会导致内存的高效。例如，限制3维或4维数组的使用，不仅使得数组元素存取算法快，占用的内存也少。因此，优化算法才是内存高效的关键。

11.3.3 I/O效率

在计算机的运行中，有很大的一部分时间在处理I/O，它的效率也影响到程序的效率。一般有两类I/O要考虑。

（1）由人支配的I/O。

（2）取决于其他设备（如磁盘或另一台计算机）的I/O。

当用户可以很容易就理解输入的内容，并且程序提供了一些手段来减少用户的文字输入量时，那么用户提供的输入和产生的输出对用户来说就是高效的。对其他硬件的I/O效率牵涉到对硬件的直接操作问题，已经超出了本书的范围。但是，从编码（和详细设计）角度，仍然可以从以下几个方面来提高I/O效率。

（1）I/O要求的数量应当减至最小，比如将读写文件的功能合并，尽量一次完成。

（2）所有I/O应当缓存，以减少过多中断次数。

（3）对于辅存（如磁盘），应当选择和使用最简单的可接受的存取方法。

（4）辅存设备的I/O，应当是块状的。

正如上面所提到的，I/O在设计阶段就确定了方式，方式确定也就决定了效率。所以，上面给出的提高I/O效率的原则，既适用于软件工程的设计阶段，也适用于编码阶段。

11.4 冗余编程

冗余（redundancy）是改善系统可靠性的一种重要技术。广义地说，冗余是指所有对于实现系统规定功能来说是多余的那部分资源，包括硬件、软件、信息和时间。比如对于一个系统，提供两套或更多的硬件，使之与原始系统并行工作。这种方式称为并行冗余，也称热备用或主动式冗余。在另外一种情况下，如果提供多套的硬件资源，但是只有一套资源在运行，只有当它失效时，备用的资源才开始运行。该方式称备用冗余，也称冷备用或被动式冗余。

在单个元件的可靠性不提高的情况下，使用冗余技术可以大大提高系统运行的可靠性。比如假设有两个平行工作的元件。单个元件工作1000小时的可靠性为0.8，即它在同一时间间隔中发生故障的概率为0.2。只有当两个零件都失效时系统才会失效，如果两个元件是相互独立的，则系统故障的概率为0.2×0.2=0.04。这时该并行系统的可靠性是0.96。可靠性提高了20%。

但是对于软件，就不能简单地照搬硬件冗余的情况。若想通过采用两个程序文本相同的程序在计算机上运行来实现软件冗余，将达不到软件冗余的目的。因为在两台计算机上程序如果是一样的，则一个软件上的任何错误都会在另一个软件上出现。要想采用冗余软件，就必须设计出两个功能相同，但源程序不同的程序。

在设计冗余软件时，不仅采用不同的算法和设计来实现同一个计算，而且编程人员也应该不同。假设要计算二次方程的实根，则可以在一个程序中使用二次求根公式，而在第二个程序中采用牛顿-拉菲逊（Newton-Raphson）数值逼近法。如果两个结果在额定的“计算误差”范围内是一致的，则可以采用任何一个结果或两个结果的平均值作为答案，并且打印出来。若结果不一致，而且不知道哪一个正确，则可采用错误检测系统来纠正。如果采用三种以上的计算，则可采纳多数答案。这种技术称为逻辑或多数表决。

在冗余编程的费用上，把一个程序制成两个冗余的程序，开发费用似乎应为编写单个程序的两倍，但实际费用可能还不到1.5倍。这是因为软件的描述、设计和大部分测试以及文档编制的费用是两个程序共享的。冗余编程引起的副作用是由于文本增加而带来的存储空间的增加，以及运行时间的延长。为此可以采用海量存储器和覆盖技术，并仅仅在关键部分采用冗余计算，这样可以使成本减到最小。

11.5 软件容错技术

提高软件质量和可靠性的技术大致可分为两类，一类是避开错误（fault–avoidance）技术，即在开发的过程中不让差错潜入软件的技术；另一类是容错（fault–tolerance）技术，即对某些无法避开的差错，使其影响减至最小的技术。避开错误技术主要体现在提高软件的质量管理，也就是软件工程中所讨论的先进的软件分析和开发技术以及管理技术。

但是，无论使用多么高明的避开错误技术，并且在理论上无法证明程序的正确性，无法做到完美无缺（zero–defect）和绝无错误（error–free），这就需要采用容错技术以使错误发生时不影响系统的特性，或使错误发生时对用户的影响限制在某些容许的范围内。特别是在一些要求高可靠性、高稳定性的系统中，例如原子能发电控制系统、飞机导航控制系统、医院疾病诊断系统、银行网络系统等等，都非常重视应用容错技术。

项目	说明
算法模型化	将可以保证正确实现需求规格说明的算法模型化
模拟模型化	为保证在一定资源条件下预定性能的实现，将软件运行时间、内存使用量、执行控制等模型化
程序正确性证明	使用形式符号及数学方法，证明程序的正确性
N个版本的程序设计法	由N个独立的软件项目组同时开发同一需求规格说明的软件，从N个版本的执行结果的不同点出发，寻求整体的一致性
容错设计	使软件具有抗故障的功能
软件风险分析及故障树分析	从设计或编码的结构出发，追踪软件开发过程中潜入系统差错的原因
划分接口的规格说明	在设计的各个步骤，使用规范的接口规格说明，经验证划分接口的实现可能性和完全性能
可靠性模型	使用软件可靠性模型，从软件故障发生频度出发预测可靠性

表11-1 软件高可靠性技术

11.5.1 容错软件

什么是容错软件？没有一个准确的定义，从不同的角度考察，归纳起来，有以下四种定义：

（1）规定功能的软件，如果在一定程度上对自身错误（软件错误）具有屏蔽能力，则称此软件为具有容错功能的软件，即容错软件。

（2）规定功能的软件，如果在一定程度上能从错误状态自动恢复到正常状态，则称之为容错软件。

（3）规定功能的软件，程序存在错误而且发生时，仍然能在一定程度上完成预期的功能，则把该软件称为容错软件。

（4）规定功能的软件，如果在一定程度上具有容错的能力，则称之为容错软件。

以上四个定义在描述上各有侧重点，但在以下三个方面是共同的：

（1）容错的对象是一个规定功能的软件，这些功能是由需求规格说明定义的。容错是为了保证当错误存在并且发生时，能维持这些功能。

（2）容错的能力总是有一定限度的。这是由于软件错误一般是不可预见的，输入信息的构成又是极为复杂的。因此，即使是容错软件也不能解决所有的错误。

（3）容错软件由于自身存在错误而在运行中出错时，应能屏蔽这一错误，对其进行处理以避免失效。通常这一功能是通过错误检测算法、错误恢复算法，并调动软件冗余备份来实现的。

作为软件的冗余备份，并不要求该软件的全部功能都有冗余备份，它可以是某些功能块、子程序或程序段。这些冗余备份和检测程序、恢复程序一起统称为容错资源。容错软件就是由实现规定功能的常规软件和容错资源组成的。常规软件是主体，容错部分的功能只是提高为了可靠性。

11.5.2 容错的一般方法

实现容错计算的主要手段是冗余。由于加入了冗余资源，有可能使系统的可靠性得到较大的提高。按实现冗余的类型来分，通常冗余技术分为四类。

1. 结构冗余
2. 信息冗余
3. 时间冗余
4. 冗余附加技术

1. 结构冗余

结构冗余是最常用的冗余技术。按其工作方式，又有静态、动态和混合冗余三种。

1）静态冗余

静态冗余通过冗余结果的表决和比较来屏蔽系统中出现的错误。其中三模冗余TMR（Triple Modulor Redundancy）和多模冗余最常用。如图11-1所示是三模（TMR）表决系统的逻辑结构。图中U是一个表决器，可以由硬件或软件来实现。

其表决输出为：

$$U=(U1 \wedge u2) \vee (U2 \wedge U3) \vee (U1 \wedge U3)$$

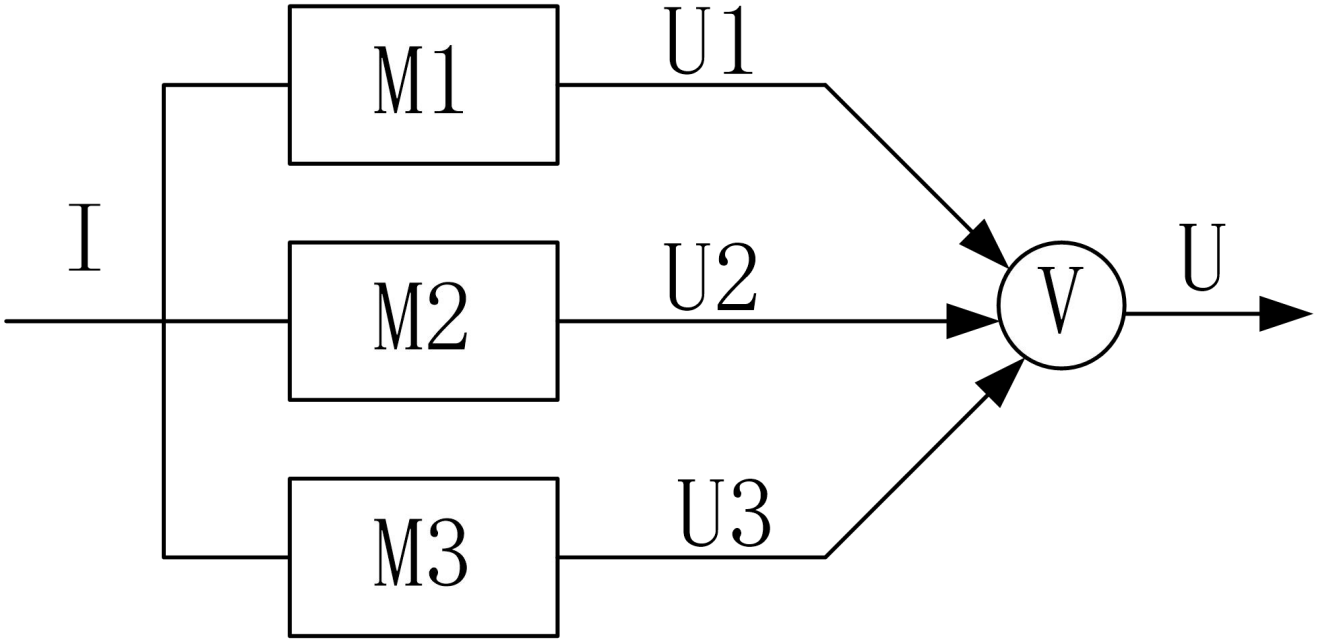


图11-1 三模（TMR）表决系统的逻辑结构

其中的U1、U2和U3分别是三个功能相同但由不同的人采用不同的方法开发出来的冗余模块的运行结果。通过上式的表决，以多数结果为系统的最终结果。这样，错误的模块的运行结果将被“屏蔽”。因为在该技术中，无需对错误进行处理，在运行时也不必进行模块的切换，故称为静态容错。

2）动态冗余

动态冗余的特点是在系统运行出错后才运行冗余模块。依靠存储多个模块，当检测到工作的当前模块出现错误时，就有一个备用的模块来顶替它并重新运行。因此动态冗余的实现是系统的检测、模块切换和恢复过程。和静态冗余的特点相比，动态冗余的结构的冗余成分是逐个运行的，称其为动态冗余。典型的动态冗余的逻辑结构如图11-2所示。

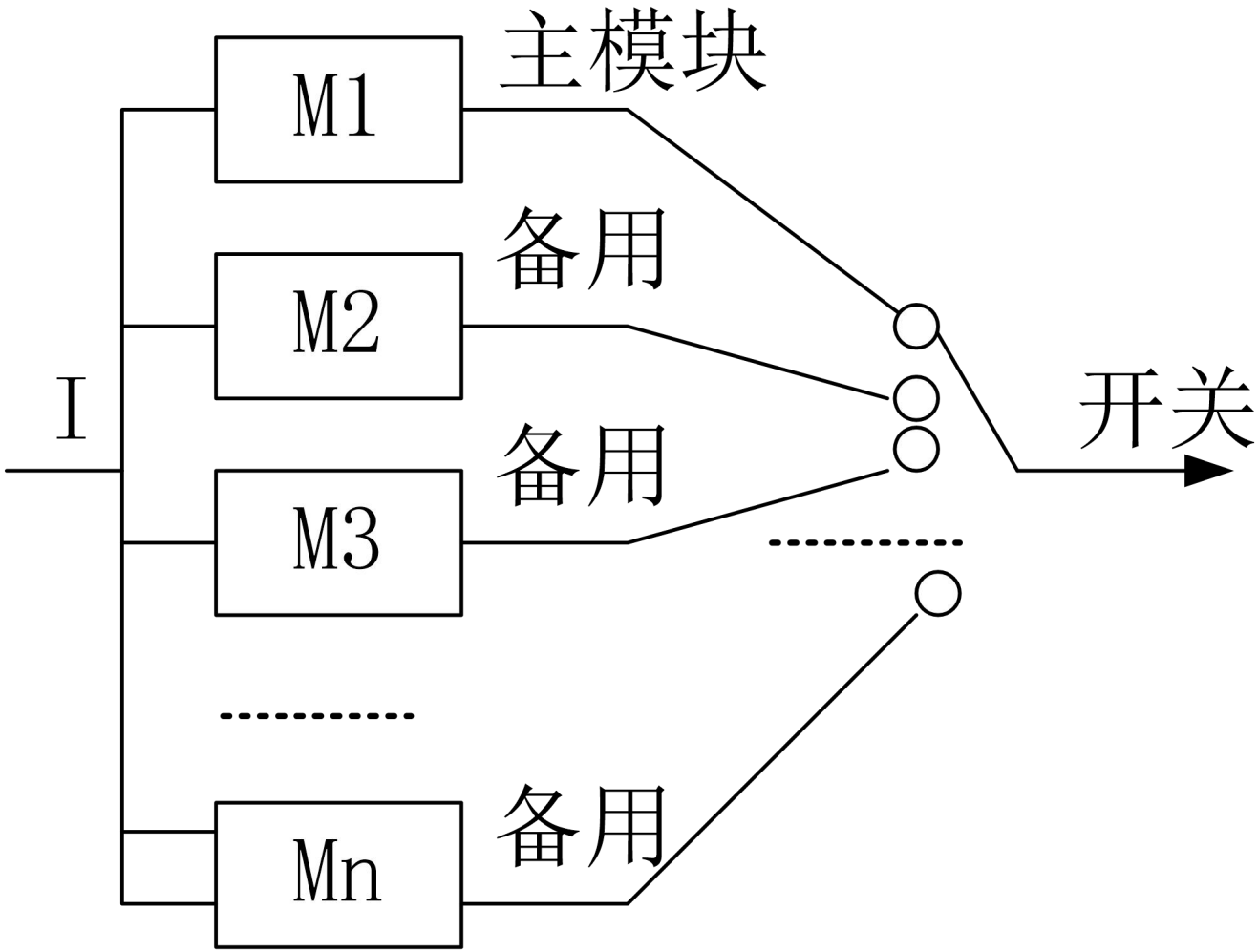


图11-2 动态冗余的逻辑结构

图中M1，M2，……，Mn是n个具有功能相同，但独立设计的不同模块。其中M1称为主模块，M2~Mn称为备用模块。当M1出错时，由M2顶替M1工作，M3~Mn又成为M2的备用模块，依次类推。只有所有n个模块相继都出错后，系统才会失效。

每当一个出错模块被其备用模块顶替后，冗余系统相当于进行了一次重构。如果各备用模块与主模块同时工作，只是其结果不影响主模块的结果，叫做热备系统；如果冗余模块不工作，当主模块出错后才开始运行，叫做冷备系统。在热备系统中备用模块在待机过程中其失效率可视为0。

3）混合冗余

结合静态冗余和动态冗余的长处，就可以得到混合冗余结构了。通常用H(N，K)来表示，其逻辑结构如图11-3所示，总共有N个模块数，从M1到MK模块组成了静态冗余结构的表决模块，而N-K个模块则作为动态冗余结构中的备份模块。

在运行中，当参与表决的K个模块中（通常K≥3）有一个模块出错时，就有一个备份模块代替该模块参与静态的表决，维持静态冗余系统的完整。混合冗余系统比N重单纯的静态系统或动态冗余系统的可靠性都高，同时系统对错误检测要求不高。

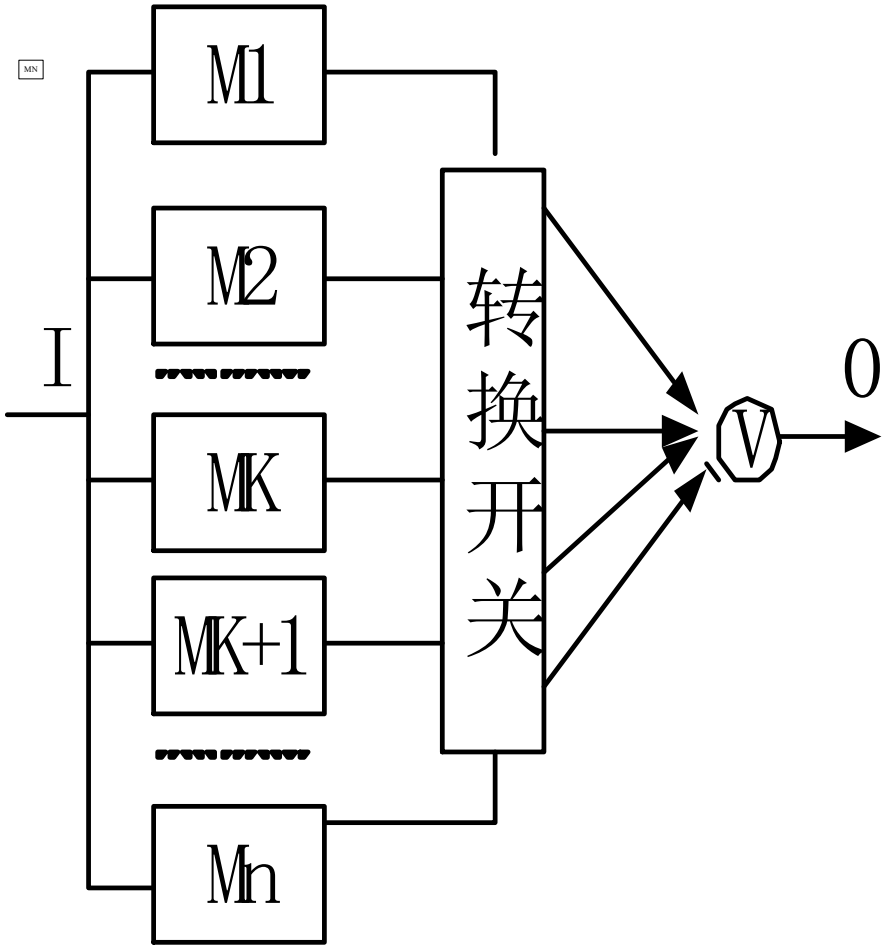


图11-3 H(N，K)系统结构

以上三种结构冗余形式都涉及到信息的表决、错误的检测、切换和切换后的重新启动等问题，各有特点，在具体的应用时，需要根据系统的背景和容错的要求，采用不同的技术手段来解决。

2. 信息冗余

信息冗余是保证信息在传输过程中不出错的重要技术方法。在通信和计算机系统中，信息常以编码的形式出现。在传输过程中很容易受到干扰而出错，现在常用的技术是采用奇偶码、循环校验码等就可以发现甚至纠正这些错误。冗余码的长度远远超过不考虑误差校正时的码长，增加了计算量和信道占用的时间。

3. 时间冗余

主要是利用对指令的重复执行（指令复执）或程序的重复运行（程序复算）来消除错误带来的影响。其运行过程的描述如图11-4所示。图中Ri和Rj为源地址，Rk为目的地址，θ为操作码，CP为指令计数器。即Ri和Rj的内容经过θ运算后，其结果送入Rk中。如果错误检测程序提出有错，并发出信号，则指令计数器的内容减1，重新执行该指令以期消除错误。如果达到了此目的，即恢复成功，则指令计数器的内容加1，继续执行下一条指令。

如果恢复不成功，则处理办法是发出中断，转入错误处理程序，或对程序进行复算，或重新组合系统，或放弃程序处理。

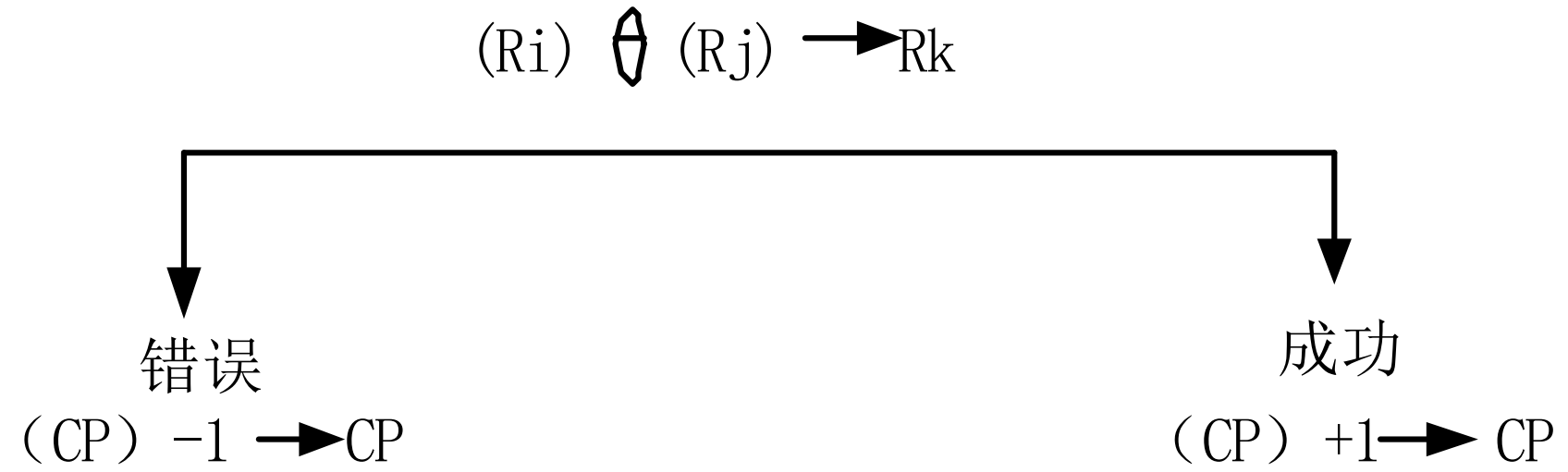


图11-4 指令复执过程示意图

4. 冗余附加技术

在实现上述的冗余技术时，实际上是对硬件、程序、指令、数据等资源进行冗余储备。所有的这些冗余资源和技术统称为冗余附加技术。在没有容错要求的系统中，它们是不需要的；但在容错系统中，它们是必不可少的。按不同的目的，实现冗余附加技术重点不同。

以屏蔽硬件错误为目的的容错技术中，冗余附加技术包括：

（1）关键程序和数据的冗余存储和调用。

（2）检测、表决、切换、重构、纠错和复算的实现。

由于硬件出错对软件可能带来破坏作用，例如，导致进程混乱或数据丢失等。因此，对它们做预防性的冗余存储十分必要。

在屏蔽软件错误的容错系统中，冗余附加件的构成则不同。它包括：

（1）各自独立设计的功能相同的冗余备份程序的存储及调用。

（2）错误检测程序和错误恢复程序。

（3）为实现容错软件所需的固化程序。

这说明容错软件也需要硬件资源的支持。容错消耗了资源，但换来对系统正确运行的保护。这与那种由于设计不当而造成资源浪费的冗余不同。

11.5.3 容错软件的设计过程

容错系统的设计过程如图11-6所示，它包括以下设计步骤：

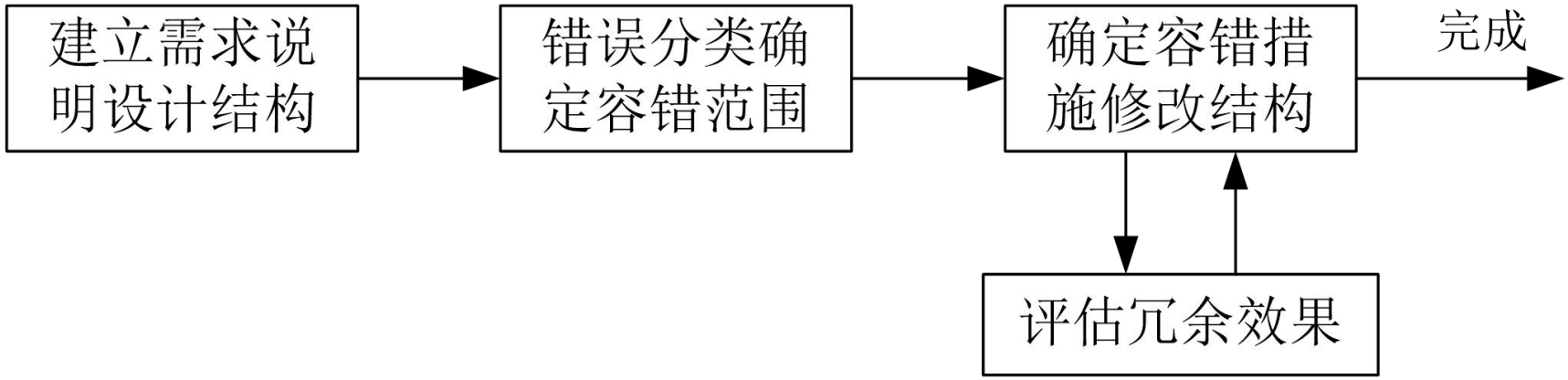


图11-6 容错系统的设计过程

（1）首先要按设计任务要求进行常规设计，尽量保证设计的正确。为达到此目的，要求合理地组织软件的开发，使用先进的方法和工具，严格地实行质量管理等一系列措施来避免引入设计错误。需要注意，常规设计得到的非容错结构，正是容错系统的构成的基础。在结构冗余中，不论是主模块还是备用模块的设计和实现，都要在费用许可的条件下，用调试的方法尽可能提高可靠性。不能忽视常规设计和实现的质量，只把目光盯住采用容错手段实现高可靠性。

（2）根据系统的工作环境，对可能出现的错误分类，确定实现容错的范围。对可能发生的错误进行正确的判断和分类，是有效地实现容错的关键。例如，对于硬件的瞬时错误，可以采用指令复执和程序复算；对于永久错误，则需要采用备份替换或者系统重构。如果判断不正确，就达不到应有的容错效果和造成资源的浪费。对于软件来说，只有最大限度地弄清错误发生和暴露的规律，才能正确地判断和分类，实现成功地容错。这种规律的掌握，要求深入调查和分析系统设计和运行背景。

（3）按照“成本-效益”最优的原则，选用某种冗余手段（结构、信息、时间）来实现对各类错误的屏蔽。对于事先难以预料的错误虽然不一定都能屏蔽，但应尽量考虑适当的对策，最后形成完整的冗余体系。

（4）分析或验证上述冗余结构的容错效果。如果效果没有达到预期的程度，则应重新进行冗余结构设计。如此反复，直到有一个满意的结果为止。

11.5.4 软件的容错系统结构

在实现容错系统的时候，需要多方面技术的支持。除了上述的冗余附加技术以外，还有错误检测、恢复、错误隔离、破坏估计和继续服务等技术。这些就不再介绍。可把这些技术根据可靠性的要求综合起来应用，使其形成一个整体，实现容错功能。不同的结构，其在可靠性上的得益是不同的。高可靠性的获得取决于系统结构。系统结构是否合理，主要看其是否满足工程上的要求和较好的成本-效益比。

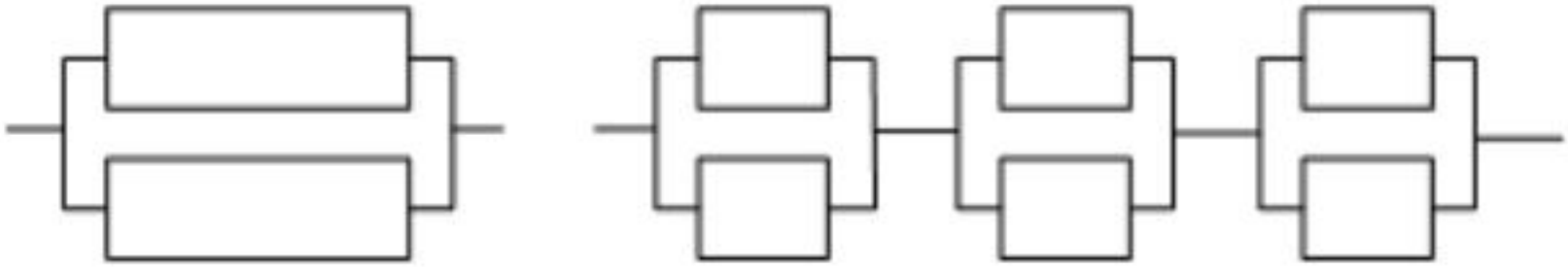
1. 最小冗余单元

在硬件系统的可靠性中，一个系统可分为子系统、模块、部件和元件。所谓元件，是指一项具有独立个性的产品，如一个电阻、一个电容或一块集成电路等。电路中的一条引线、一个焊点，虽然能够因其失效而导致整个系统失效，但并不作为最小冗余单元。

在软件中，最小冗余单元是指其功能可以明确定义的最小程序段。一个字符、一条语句的错误会导致软件的失效，但如果没有独立的功能，就不是最小冗余单元。

2. 容错域

一个系统在具体设计其冗余结构时，至少有两种可供选择的方案，如图11-7所示。



（a）大容错域　　（b）小容错域

图11-7 大小不同的容错域

图11-7（a）是将整个系统与另一个功能完全相同的备份一起构成冗余系统来实现容错。图11-7（b）所示的是把系统按功能划分为小段，每一段是由一个主模块和一个冗余备份模块构成一个小的冗余子系统。前则是在大的范围内完成容错功能，后者要求系统在每一个小范围内就完成容错功能，而这就是容错域的大小问题。

假设图11-7（a）所示系统的两个备份相同且均由n个段组成。每一个备份的可靠度可利用串行模型的可靠度计算公式求得：

$R=R1\cdot R2\cdot R3\cdot\ldots\cdot Rn$

在不考虑容错支持部分（冗余附加件）影响的条件下，容错系统的可靠度可利用并行模型的可靠度计算公式求得：

$$R_{FTA}=1-(1-R)\cdot(1-R)$$
$$=2R-R\cdot R$$
$$=R\cdot(2-R)$$
$$=R_1\cdot R_2\cdot R_3\cdot\ldots\cdot R_n\cdot(2-R_1\cdot R_2\cdot R_3\cdot\ldots\cdot R_n) \quad (11—1)$$

图11-7（b）所示系统中每一个冗余子系统的可靠度按上述并行模型的方法可得：

$RFT1=R1\cdot(2-R1)$

$RFT2=R2\cdot(2-R2)$

…

$RFTn=Rn\cdot(2-Rn)$

整个系统按串行模型求出的可靠度为：

$$R_{FTB}=R_1\cdot R_2\cdot\ldots\cdot R_n\cdot(2-R_1)\cdot(2-R_2)\cdot\ldots\cdot(2-R_n) \quad (11—2)$$

由于可靠度的值是一个小于1的值，比较(11—1)和(11—2)可知：

$$R_{FTB}\ R_{FTA} \quad (11—3)$$

由此说明容错域小的系统比容错域大的系统可靠度要大。通常容错域的选择在硬件中可以以功能级为基础加以考虑。例如存储器、电源等。在软件中，容错的得益取决于各冗余备份的设计独立性。相互独立的程序，即使都可能存在错误，但其他程序受其影响的可能性却相对较小。当主备份出错时，利用备份程序就可能得到正确的结果。相反，如果软件之间的独立性差，那么无论把程序划分为多少个容错域也不会提高系统的可靠性。

3. 软件的容错系统结构举例

目前常用的软件容错系统结构有两种：

1）多版本程序（Multi-version Program）结构

该程序系统是由2个以上满足同一需求规格说明的版本构成。这里的n（n≥2）个版本是让n个程序开发组各自独立地开发满足同一个需求规格说明的程序。由于这些开发组使用的设计、编码和测试方法不会完全相同，只是接口要求一致，所以这n个版本内潜入的错误也不会相同。换言之，这几个独立开发的软件对同一输入处理的结果，不可能发生同样的错误现象。

把这些同一功能的不同版本的程序（多为子系统级或模块级）并行联结到系统中，或者分别在n个计算机上并行运行，就构成一种冗余并行模型。在程序运行当中，这些不同版本的程序模块并行地执行同一功能，对它们产生的结果，经过一次多数表决，作为执行的结果。如图11-8所示。

由于各个程序在运行中多少都会存在一点误差，因此通常采用“非精确表决”算法。可在一个设定的允许误差范围内进行比较和抉择，或者根据对各程序模块已做的可靠性推断，对不同程序模块的运算结果加权，使其在表决中起更大的决定作用。

2）恢复块（Recovery Block）结构

程序可划分为若干个块（块可以是过程、子程序或程序段）。程序是块的集合。

给要求做容错处理的块（叫做基本块）提供备份块（即独立设计的相应冗余备份）、附加的错误检测、恢复措施和接受测试过程后，就把一般的块变成了恢复块。其结构如图11-9所示。

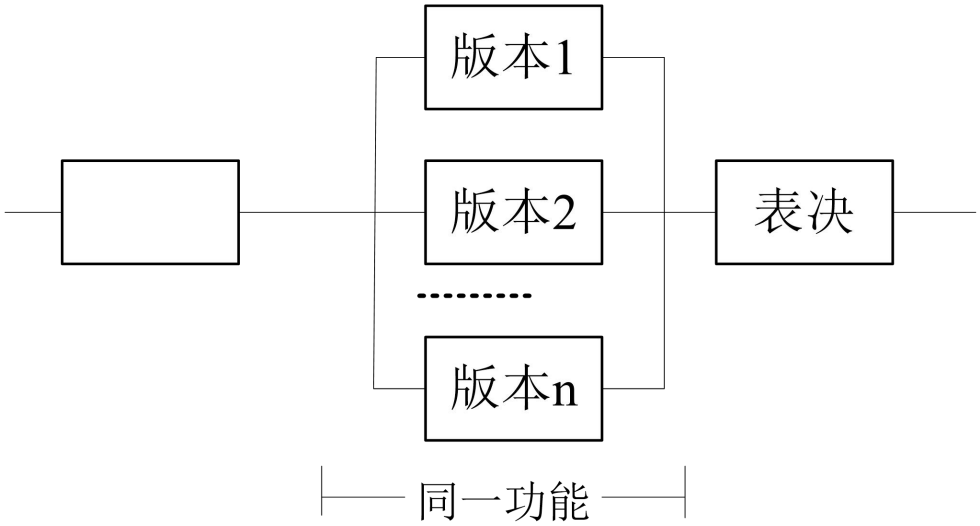


图11-8
多版本程序的示意图

```
Ensure    接受测试
          By 基本块
          Else By 备份块 1
          ...
          Else By 备份块 n
Else      错误
```

图11-9
恢复块的结构

从图中可知，恢复块由基本块、备份块和接受测试等部分组成。各备份块的设计独立于基本块且相互独立。备份块按要求可以与基本块具有同样的功能，也可以设计成基本块的降级，但这种情况的接受测试与基本块不同。恢复块的工作方式是一种冗余方式。即运行时首先对基本块进行接受测试，如果接受测试成功则执行后续工作，否则用备份块替代基本块，重新开始运行。其工作方式如图11-10所示。

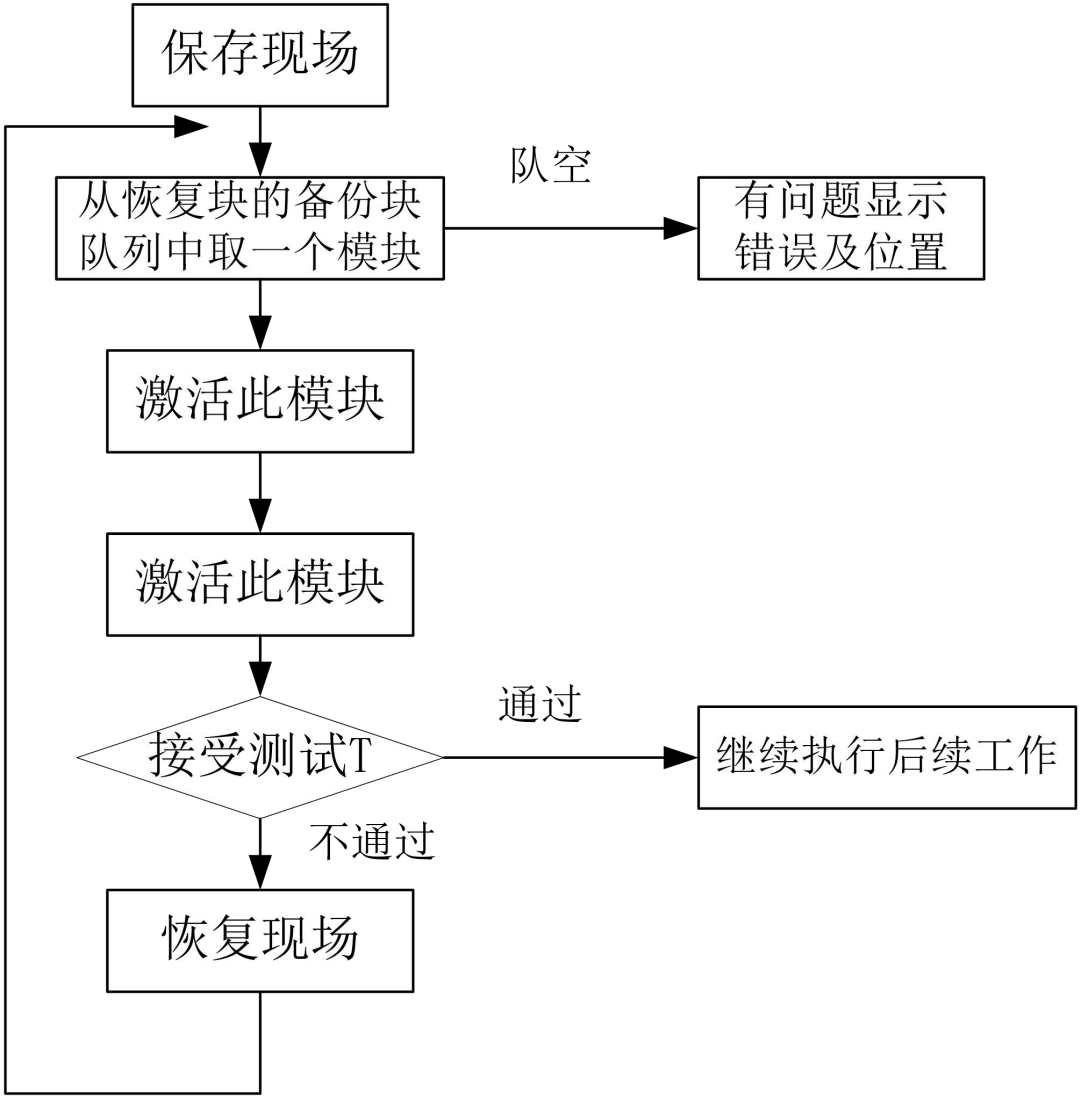


图11-10 恢复块的工作方式

接受测试T是对模块执行结果进行评价估算的确认性检验。它可用于检验现在是否有错误。如果检查通过，才可以继续执行后面的工作；如果检查不通过，则系统恢复到这个模块执行以前的状态，再换一个具有相同功能的不同版本的另一个备份模块执行。接受测试T的两个设计标准是：

（1）能检测实际运行与要求的偏差。

（2）防止非安全的输出。

小结

本章主要介绍了软件实现。软件实现就是“编码”。其实在软件开发的需求阶段就要考虑软件实现所用的程序设计语言了。计算机发展到今天，出现了众多的编程语言，有的昙花一现，有的长久不衰。但是不同的程序设计语言却有自己不同的特点，有的适合面向硬件，有的适合实时性系统，有的适合开发数据库。因此要根据系统的需求来选择程序开发语言。

小结

在编写程序时，应该以一种公认的、易于理解的格式书写程序，包括嵌套要缩进，不要在一行上书写多个语句等。良好的编程风格带来的是程序的可读性。

在程序设计时，还要考虑程序的效率，包括对硬件资源的使用效率，算法的效率等。但是不能为了追求效率而损失程序的可读性和清晰性。

小结

冗余编程的目的是改善系统的性能。冗余的代码对提高系统的可靠性有显著的影响。

软件的容错技术也是提高系统的可靠性的。容错软件是使错误发生时不影响系统的特性，或使错误发生时对用户的影响限制在某些容许的范围内的技术。结构冗余、信息冗余和时间冗余是常用的容错技术。

容错软件一般是由错误检查机构和错误恢复部分组成的。

谢谢