第3章

软件视图和文档风格

3.1 软件视图的概念和类型

现在，读者应该已经知道软件文档能够起到多种作用：软件需求的描述、软件开发者的任务说明、系统规划和分析的基础、系统实现规则、代码自动生成的规范、系统理解，以及项目投资回收的分析或项目开发计划，等等。

那么，什么是软件视图？它的作用和意义又是什么？回答这个问题主要取决于你的目标！

所谓软件文档，一般以文字、图、表等方式记录目标系统的定义、规划、分析、研究、设计、开发、应用等各个阶段的设计思想和研究开发成果。因此，软件文档为了全面、完整、系统地描述这些成果，许多时候更以“图”的形式来直接、形象地表达一些文字难以简洁描述的内容。而这些“图”为了更完整、精确地反映被表达对象其内部的逻辑结构，以及外部各“图”之间的逻辑关系，以若干“图”来构造一组系统元素及其相互关系的方式，构成所谓软件视图，来形象、直观地反映软件设计的结果。

视图是对一组系统元素及其关系的描述。

软件视图是软件的一种表达形式，或者是有关软件的一种报告。无论视图是否可以被人看到，作为软件的重要过渡表示，视图的涉众通常都希望能够看懂或者说是读懂视图。视图信息是视图中出现的特定信息，或者是根据视图中信息分解而得到的知识构成的信息库。所以，软件视图是软件文档的一个组成部分。

因此，软件编档的一个十分重要的工作就是为相关视图编档，然后和添加的适当的文字、表等内容一起形成完整的软件文档。软件文档的构成如图 3.1 所示。

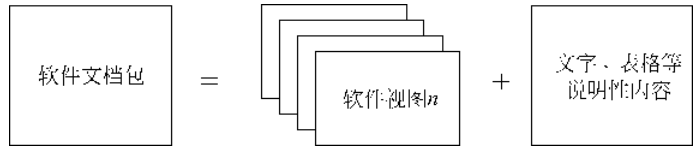

图 3.1 软件文档包的构成

从图中可以看出，软件文档包由一个或多个视图文档加上说明文档，来向读者介绍该文档包的内容，并引导读者阅读该文档包的文档组成。

通常，这类带有视图的软件文档包括了用来描述软件开发项目的技术性或开发性资料，

如系统分析文档、系统构架文档、系统设计文档等。

软件视图的实例包括软件规格说明、数据流图(DFD)、源程序、度量值、根据静态源程序导出的报告,以及表明软件性能的测试数据等。图 3.2 描述了一些软件视图的示例,其中含有视图信息。

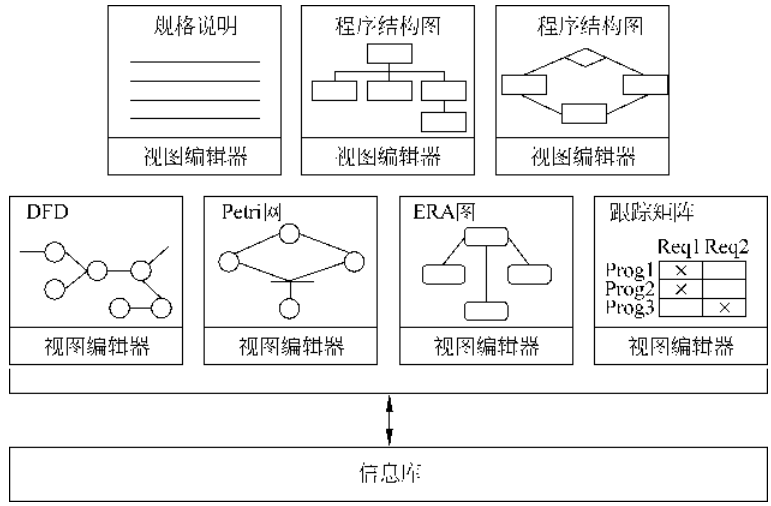

图 3.2　软件视图的示例

图 3.2 中,信息库是有关软件的信息仓库。有三种途径可以把有关软件的信息存入库中:

- 分解软件为对象和关系。
- 逐步得到对象和关系,借助工具将其送入信息库。
- 从其他信息库引入信息。

所以,当软件视图得到工具支持时,就成为视图编辑器,可以用来支持加入、浏览和变更视图信息。

另外,不同的视图及视图种类会在不同程度上表现出不同的质量属性。因此,你和软件文档涉众在系统开发过程中,最为关心的质量属性将影响到在编制软件视图时对视图的选择。例如,本章稍后介绍到的分层视图就能反映系统的可移植性,而部署视图则可以推断系统的性能和可靠性等。

另外,不同种类的视图也被用来支持不同的目标和用途。当软件编档需要选择视图时,应该首先明确文档的预期使用。因为,不同种类的视图会突出不同的系统元素和(或)关系。

但是,许多软件开发项目的设计者,会错误地将单一的系统划分方式强加给多个组件领域。例如,将线程等同于对象,将对象等同于模块,而后又将模块等同于文件。这种对系统的划分方法引起了混乱,因此,从一开始实际上就注定了它是会失败的,最终必然是需要作出重大调整的。然而,在软件开发领域,尤其是在软件系统分析、规划和设计阶段,初始意图的缺失、不完整,甚至错误,其对系统结构、功能、构架、质量等所造成的损害,一般是很难修复的。所以,不正确的方法无疑会导致一些系统开发的问题,偶尔更会出现最终产品问题。

软件视图作为展示软件结构、构架的有效工具，就单个视图而言，其实质是压缩软件系统当前表达的内容中不需要的信息，这样就使得每个视图总是只展示系统某一两个方面。而系统更多的信息，就需要通过多个分散的视图来进行观察和了解。这样做的好处是，每个视图仅强调系统的某个方面，而忽视或不予强调系统的其他方面，以便有助于更好地突出当前问题。换句话说，多个视图是无法充分记录或展示系统完整的结构的。而要达到这一目的，就需要具备一个能够记录系统各种结构形态的完整的视图集，以及对这些视图进行补充说明的必要信息。

视图文档包含以下几部分：

- 描述主要元素和视图关系的主要表示，一般会用图形表示。
- 说明和定义视图所展示的系统元素，并列出其属性元素的目录。
- 该系统元素的接口和行为规范。
- 对所有针对体系结构进行剪裁的现有内部机制进行说明的可变性指南。
- 基本原理和设计信息。

接口是指两个独立的实体相互接触、交互或通信的边界。

并不是所有文档都可以用来介绍、描述或说明所有各类视图的。以下所列是适用于所有视图的文档：

- 对整个文档包的介绍，包括帮助涉众迅速查找所需信息的读者指南。
- 描述视图之间和视图与整个系统之间如何相互联系的信息。
- 针对整个软件体系结构的限制和基本原理。
- 有效维护整个文档包所需的管理信息。

事实上，和建筑结构类似的是，一个稍微复杂一些的软件系统，其体系结构就不能仅采用一种视图来给出描述，而是需要有多种系统视图；且每种视图仅强调一些确定的体系结构特征，而这些特征则适用于描述不同的体系结构，适用于不同的涉众或不同的用途。

这是由于，软件是由许多不同的结构组成的。这些结构有可能是特殊的，比如与操作系统存在相当特殊的关系，规定什么进程拥有哪个内存段。而更多的其他结构则通常更具一般性，其应用当然也更为广泛。这些“一般”的结构包括：

- 模块结构——单元是工作任务，关系如“与……同一逻辑结构的部分，或与……共享同一逻辑结构中的部分逻辑结构”；
- 用途结构——单元是程序，关系如“依赖……的正确性”；
- 进程结构——单元是进程，关系如“向……提供计算服务”。

通过对大量用于描述软件体系结构的视图进行分析，Rational 软件公司的 Kruchten 对 4 种主要的软件结构描述视图进行了归纳性的描述，在构建系统的过程中，这些视图还能与第五种所谓突出的视图一起发挥极为重要的作用。而后者能将其他 4 种视图结合，构架所谓的 4＋1 方案视图集。

- 逻辑视图。主要支持行为需求，即描述系统应该为其最终用户提供的那些服务。在逻辑视图中，设计者将系统分解为一组取自问题域的关键性抽象概念，这些抽象概念是利用抽象、封装和继承的原则构造的对象或对象类。除了协助功能分析外，系统分解还能有助于识别系统中常见的一些机制和设计元素。
- 进程视图。能够处理并发和分布、系统完整性以及容错性。进程视图还能指定由哪

个控制线程执行在逻辑视图中确认的类的各项操作。因此，涉众可以将进程视图看成是一组由通信程序组成的独立执行逻辑网络，这些通信程序（即进程）分布在一组硬件资源中，而这些硬件资源又连接总线、局域网或广域网。

- 开发视图。描述软件开发环境中的软件模块组织。开发视图的单元是由一个或多个开发者开发的较小的软件开发块，即程序部件或子系统。开发视图支持对系统需求或开发团队工作的分配，支持对项目的计划、成本评估、进度监督，以及对软件重用、可移植性和安全性的推断。
- 物理视图。考虑系统需求，如系统可用性、可靠性、性能和可扩展性等。物理视图最终将在逻辑视图、进程视图和开发视图中确认的各种系统元素——网络、进程、任务和对象——映射到系统处理节点。

最后，再使用所谓小的重要场景——用例的实例——的子集，来说明 4 种视图元素是否能够密切配合，即“+1”视图。“+1” 视图和前 4 种视图一起，构成所谓的 4+1 方案。4+1 方案也一直被 Rational 软件公司作为统一过程（Rational Unified Process）的基础部分。

大约与此同时，西门子公司研究中心和 Herzum、Sims 两人，也分别提出了他们各自的“视图集”方案。

3.2　软件视图类型

尽管固定的视图集未必适合每个系统，但是，广泛的指导原则仍然能够帮助那些系统分析师在实施项目系统分析和设计时取得一个立足点，尤其是在进行分析和设计编档时更是如此，这样可以使得自己的描述对象有一个中心聚焦点，以便于根据系统结构建立分层或发散的开发文档。这就要求系统分析师必须同时以下述 3 种方式对自己正在分析、设计的软件系统进行考虑：

- 作为一个能够实现有效功能的单元集合，系统应该如何构造？
- 作为一个具有运行时行为和交互作用的元素集，系统又该如何构造？
- 系统如何在自己所处环境与非软件结构产生联系？

这 3 种方式代表着系统设计师在分析和设计系统时必须考虑的 3 种视角：将系统看成实现单元、运行时执行单元、从软件元素到环境结构的映射。因此，是对软件系统构造进行描述的 3 种视图类型，而文档就是记录这些内容。这 3 种视图类型分别是：

- 模块视图类型——能为系统主要实现单元编制文档服务。
- 组件和连接器（C&C）视图类型——能为系统执行单元编制文档服务。
- 分配视图类型——能为系统软件与其开发和执行环境之间的关系编制文档服务。

视图类型是指可以从特定角度定义用来描述软件系统结构的元素类型和关系类型。

3.3　软件文档风格的概念

在软件编档的过程中，特别是对一些软件系统的结构进行分析和设计的编档过程中，重复出现说明性文字文档或视图这种形式被广泛遵守，即使是在为表象可能完全不同的系统

编档时也如此。

由于这些形式出现的频度是如此之高，并且它们可能拥有其他格式所没有的独特的重要属性，或许还能代表重要的或常用的视图类型，因此，完全有必要归纳总结和了解它们，试图形成若干种重要和常用视图类型描述时常见的形式和变体，成为具有一定适用性的编档风格，以帮助确定编档的方法和规范，使软件文档能够更好地为开发和涉众服务。我们需要了解的是，在视图类型的范围内，选择风格就是要作出以下必要的选择：元素是如何受到限制的？元素是如何相互联系的？对元素的使用或配置存在什么限制？风格既然会对软件编档产生影响，因此就完全有必要对它们进行定义和讨论。

风格是对元素和关系类型的特化，包括如何使用这些元素和关系类型的一组限制条件。

具体地说，风格是文档表达的体现，也是视图类型的具体化，它能反映交互作用重复出现的模式，并独立于任何特定的系统。风格允许将专门的设计知识应用于特定类型的系统，也允许利用特定风格的工具、分析方法和实现方案支持这类系统的设计和实现。

例如，可以通过限制模块的输入信息源，来实现对模块进行一些良好的配置，这样就产生了一种所谓的分层风格——模块视图类型中的一种。这种风格可以向使用该模块的系统定向分配该层面上的可修改性、可移植性以及有目的地快速抽取有用子集的能力等属性。由于现代软件拥有层次结构，任何软件系统都会拥有不同数量的层，不同层包含不同的内容，因此针对每个不同的层可以使用的规则也不同。

再如，客户机-服务器是一种常见的系统构架形式，对其描述的编档描述方式形成了一种构架风格，即组件和连接器视图类型中的一种。这种风格中出现的描述元素主要有客户机、服务器和表达其交互作用的协议连接器。实际使用时，客户机-服务器风格能向系统分配需要的属性，如添加客户机的数据处理能力、限制客户机的网络访问量等。这样，在该风格下，不同系统将会由于所分配属性的不同而不同。如客户机、服务器等数量的不同，服务器处理能力的不同，客户机被授予的能力的不同，甚至是协议连接器中协议的不同等，都会使得系统发生很大甚至是根本的变异。

和软件文档类型及软件工程标准一样，风格也分为多种。比如，有些风格可以适用于所有的软件系统，例如分解、使用、工作任务和部属等。而其他一些风格则仅适用于某些专门的系统中，例如分层、客户机-服务器、通信进程等。这些需要系统设计师来做合适的选择。

3.4 软件文档风格指导

由于软件文档风格一旦被确定后，直接影响到软件或其部件的结构，因此可以说，选择了软件文档风格就选择了系统风格。具体来说，即需要在相关的软件文档中记录这些风格对系统结构施加的影响，如具体化和限制条件，以及风格赋予系统的属性等。

软件文档中的这一部分描述称为风格指导。风格指导既可以引用相关软件工程文献中已有的风格描述，也可以创造一种属于自己的风格。不过，如果选择创造一种属于自己的软件文档风格，就必须为之编制完整、合用、成熟、系列化、规范化的风格指导。

风格指导是对文档风格的一种描述，其中规定了设计符号集（元素和关系类型集）和使

用这些符号集的规则(布局和语义限制条件集)。

因此,如果说全体软件文档风格集合是一个系统设计和结果记录的方法集,以及在这个方法集上的语境的话,风格指导描述的就是如何在此语境下使用这个方法集。

风格作为影响软件文档编制的表达方式,具有相同的、构成所谓标准风格指导结构的同一轮廓。

1. 概述

对视图类型/文档风格为什么有益于软件编档进行了解释。同时,还描述了视图类型/文档风格将处理系统的哪一方面,以及是如何支持系统推断和系统分析的。

2. 元素、关系和特性

① 元素

元素是视图类型/文档风格固有的结构组成块。

元素描述部分将说明元素在系统结构中的作用,并提供有效的视图元素文档编制指南。

② 关系

关系能确定元素为完成系统任务而进行协作的方式。

将列举元素之间的关系名称,并提供元素如何能以及不能产生联系的规则。

③ 特性

特性是有关元素及其相关关系的附加信息。

通常,为视图编档时,需要给特性赋值,并在相关文档的编制过程中完成。例如,层的特性——模块视图类型中分层风格的元素——包括了层的名称、层所包含的软件单元,以及层所提供的虚拟机的一些性质等。由于每个层的分层视图将命名层的名称、它所包含的软件单元以及可能提供的虚拟机,因此,在为相应视图编档时,就需要根据系统原理,对视图类型/风格中将会存在的特性予以填充。

3. 支持什么和不支持什么

描述视图类型/风格内的视图支持或不支持的推断类型。将典型用户以及他们选择使用的视图进行描述,以方便设计者或编档者在使用时了解该视图类型/风格中的视图的意图或含意。

4. 表示法

介绍为视图类型/风格编档的图形/文本表示法,以方便视图类型/文档风格中视图文档的编制。不同的表示法还可以支持主要表示中各类信息的传达。

5. 与其他视图的关系

描述视图类型/风格中的视图与不同视图类型,甚至相同类型视图中不同风格的视图之间的关系。当两种风格中的视图传达了存在细微差异的相关系统信息时,由设计师确定选用的原则以及最终选用哪种风格。

该部分内容还可能包括一些对其他有关视图存在的影响系统及其涉众的警告信息,再

有就是可能包括一条有利于将这一视图类型中的视图与其他视图建立映射的建议。

6. 范例

将提供或介绍按指定风格编档的系统范例。

【案例 3.1】 一个根据上述风格指导给出的例子——客户机-服务器风格。

1. 概述

在组件和连接器(C&C)视图类型的客户机-服务器风格中,组件通过请求其他组件的服务进行交互。这种风格表示通信一般是成对的,并且通常由客户机发起。其服务器能够通过一个或更多的接口提供服务,而客户机则使用系统其他服务器提供的各种服务。系统可以仅采用一个中心服务器,或采用多个分布式服务器。

2. 元素、关系和特性

表 3.1 是遵循这一风格指导而给出的客户机-服务器风格一览表。其中,组件类型是“客户机”、“服务器”。客户机-服务器风格的主要连接器类型是用于服务调用的请求-应答连接器。服务器拥有接口,这些接口能够描述它们提供的服务。服务器能够依次通过请求其他服务器的服务来实现客户机的功能,以形成一种服务调用的分层结构。当指定给服务器的服务请求超过一项时,通常就会利用某种协议为这些请求在服务器中定序。

表 3.1 客户机-服务器风格一览表

元素	• 组件类型:请求其他组件服务的客户机和向其他组件提供服务的服务器 • 连接器类型:请求/应答,即客户机对服务器服务的非对称调用
关系	连接关系使客户机与连接器的请求角色相关联,使服务器与连接器的应答角色相关联,并确定哪些服务能由哪些客户机请求
计算模型	客户机能启动各项活动,向服务器请求所需服务,并等待这些请求的结果
特性	与 C&C 视图类型定义的特性相同,但能根据服务器精化为:可连接的客户机数量和类型,以及性能特性。如每秒可处理的事务
布局	一般没有限制。但特化可施加以下限制: • 与给定端口或角色的连接数量 • 服务器之间允许存在的关系 • 级

3. 客户机-服务器风格支持什么和不支持什么

这是一种把客户应用程序和它们使用的服务分离的系统视图。这种视图风格能将常用服务分离,从而实现对系统的理解。同时,这种风格的功能分组方法为将系统部属到硬件平台上的互操作奠定了基础。把功能划分成客户机和服务器后,即可将它们单独分配给各级,以便支持性能可扩缩性和可靠性。

客户机-服务器系统分析包括:确定系统服务器是否提供客户要求的服务,以及客户机使用服务的方式是否适当。如,是否遵守对服务调用的定序限制等。其他分析还有:依赖性分析——比如,了解系统是否能从服务故障中有效恢复;安全性——比如,确定服务器提供的信息是否被限制在拥有适当权限的客户机的范围内;性能——比如,确定系统服务器是否能保持预期服务请求的规模和速度。

4. 与其他风格的关系

与其他组件和连接器(C&C)视图类型风格类似，客户机-服务器风格也能使服务产生者和服务消费者分离。事实上，客户机-服务器风格只是通常程序编程语言中，过程/函数/方法调用的一种泛化。另一种涉及往返通信形式的风格是对等连接风格，如无中心网络中接入的拥有相等处理权限的各节点之间的连接。但是，这种连接具有对称性，这与客户机-服务器风格有所不同。

如果将客户机和服务器进行分组，并且将它们部署到分布式环境的不同位置，就可以方便形成包含 n 级的分层结构。

5. 客户机-服务器风格的范例

互联网是一种基于超文本的客户机-服务器形式的系统，它允许客户机访问分布在全球不同地理位置的 Internet 服务器上的信息。客户机访问的内容是以所谓超文本标记语言(Hypertext Markup Language，HTML)编写，由 Web 服务器利用超文本传输协议(Hypertext Transfer Protocol，HTTP)提供的多媒体信息。这里，HTTP 是请求-应答调用协议的一种形式。作为一种无状态协议，每当服务器响应客户机的请求后，客户机和服务器的连接就会自动终止。

例如，模拟客户机发出一个 HTML 页面浏览的简单请求。当客户机知道某个需要访问的服务器的身份(如该服务器的 IP 地址或域名)后，就会利用统一资源定位器(URL)来定位和访问该服务器。具体的，客户机向服务器发出一个首部报头形式的请求，该首部报头应包含指定用于请求的方法和客户机特性。其中，客户机特性应该包括用户名、浏览器和支持的文档类型。服务器收到请求后，能以客户机约定的文档类型发送数据。具体方法是，服务器以首部报头形式向客户机发送一个响应，以指定事务处理的状态和待发送数据的类型，首部报头将在数据发送前首先发送给客户机。

这个文档风格指导，同时也将是本书后续介绍的所有视图类型和风格描述中都将遵循的标准文档风格指导结构。

根据上述关于视图类型、风格和风格指导的介绍，可以明确的一点是，文档风格是视图类型的具体化，它能够反映交互作用的重复出现的模式，并独立于任何特定的系统，因此是特定系统的一种抽象。即使是在文档风格范围内，也必须明确，文档风格中的元素和关系类型与系统中的元素和关系是如何产生联系的。换言之，就视图类型与风格而言，视图就是一种与特定系统联系的风格。图 3.3 揭示的就是视图类型与风格之间的关系。

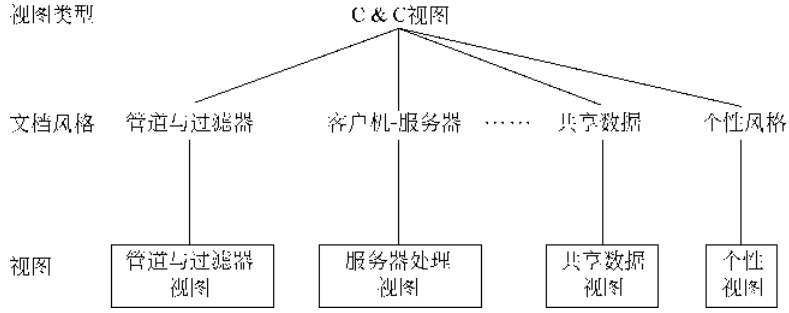

图 3.3　视图类型与风格的对应关系

3.5 模块视图及文档风格

3.5.1 模块视图类型

1. 概述

早期计算机程序的开发是没有模块概念的。模块概念最初产生于20世纪60年代、70年代，是人们为了解决所谓软件危机，提出结构化程序设计思想的基础。模块作为软件系统的最基本单元，一般具有定义良好的接口和一组精心规划的服务，以函数或过程的形式提供一个操作指令序列或算法集，实现了全部或部分的信息隐蔽。现代软件开发技术中，如在面向对象的开发语言和建模表示法(如UML)中，已经广泛采用了这种思想方法。

随着模块概念的建立和逐渐清晰，模块化、层次结构、信息隐蔽、抽象、内聚和耦合等软件工程思想也日益被人们所接受，模块成为软件系统中最小的实现单元。

实际上，软件开发人员经常使用的术语——模块，泛指各种软件结构，如编程语言中的过程、函数、标准库函数、对象类或者一般性的源代码单元组合等。为了方便描述和理解，统一模块这个概念的意思，我们给出一个更为概括的关于模块的定义：**模块**是能够提供内聚功能集合的软件实现单元。

对系统模块或主要实现单元设计过程的描述和设计结果的记录，是系统模块文档的主要内容。系统模块文档的主要职责就是枚举并描述系统的主要实现单元或模块及其内部实现逻辑，以及这些单元相互之间的关系。对这些单元或模块内容进行描述的结果称为**模块视图**。

事实上，这些模块视图除了描述单元或模块的实现外，还可以用于软件系统及模块的构架、结构建立的分析基础，可以用于与开发项目相关人员的培训，文档涉众之间的交流，以及作为开展对项目进行管理、评审和审核等工作的重要依据之一。

目前，系统软件分解成可管理单元的方式仍然是重要的系统结构形式之一。至少，这样的分解方式在目前看来，是能够帮助确定系统是如何划分成可分离的部分的，每一部分能对与其有联系的其他部分提供什么样的服务，以及这些部分是如何集聚成更大的功能或逻辑集合体，直至最终集成为一个完整系统的。模块化的选择使得系统具有支持可修改性、可移植性和可重用的能力。

当希望编制一个完整的软件文档时，应该至少采用一个模块视图类型视图。否则，无法想象没有模块的系统是一个怎样的系统。因此，在为软件文档包制定计划时，必须至少包含一个模块视图类型的视图。这也是系统开发文档，尤其是模块文档质量评审的一个关注点。

模块视图类型的最一般形式的常用文档风格有4种：

- 分解风格。表示模块之间的包容关系。
- 使用风格。表示模块之间的功能依赖关系。
- 泛化风格。表示模块之间的特化关系。
- 分层风格。表示模块之间受限制的“允许使用”关系。

每一种风格都能对基本模块视图类型进行限制。也许，还会添加某些元素和关系类型的专有形式。

2. 模块视图类型的元素、关系和特性

表 3.2 是根据所谓标准文档风格指导结构，对有关模块视图类型的元素、关系和特性进行总结的结果。

表 3.2　模块视图类型总结

元素	模块视图的元素就是模块，是一种能提供内聚功能单元的软件实现单元
关系	模块视图的关系主要有“部分关系”、“依赖关系”或“特化关系” • 部分关系。定义了子模块 A(部分模块或子模块)和聚集模块 B(整体模块或父模块)之间的部分/整体关系 • 依赖关系。定义了模块 A 和模块 B 之间的依赖性关系。特定的模块风格将详细描述依赖性的含义 • 特化关系。定义了较为特殊的模块(子模块 A)和较为一般的模块(父模块 B)之间的泛化关系中的一种
元素特性	模块特性包括以下几种： • 模块必须遵守的命名空间规则 • 模块的责任和义务 • 模块接口可以精确地确定模块在系统中的作用 • 实现信息。模块应有的可执行代码指令集
关系特性	• 部分关系：拥有相关的可见性特性，该特性确定子模块在聚集模块之外是否可见 • 依赖关系：拥有定义的约束条件，可以深入规定两个模块之间的依赖特性 • 特化关系：拥有实现特性，它表示较为特殊的模块(子模块 A)继承了较为一般的模块(父模块 B)的实现方案，但 A 不保证支持 B 的接口，因此父模块未必具有可置换性
布局	模块视图类型没有可继承的布局约束条件

(1) 元素

这里所指的元素，就是模块。

模块特征的描述可以通过枚举该模块的一个责任和义务集合来实现，它们反映的是该模块最重要的特性。这个“责任和义务”集合，涉及了软件单元可能提供的各种功能。

由于模块粒度可以有不同的定量度量，因此模块既能聚集，也可以分解。不同模块视图能根据其所基于的不同文档风格标准来确定其不同的模块集，并根据系统构架或设计结构确定的模块粒度度量值，对它们进行聚集或分解。如，分层风格和分解风格就具有不同的关系，这将直接影响它们聚集和分解行为的方式。

(2) 关系

模块视图的关系主要有：部分关系、依赖关系和特化关系。这些关系规定了模块为完成系统任务而进行协作的方式。模块视图类型所具有的关系，具体描述见表 3.2。

(3) 特性

模块特性在编档时，将作为模块视图支持文档的组成部分而编入系统文档。而模块集是否具备某些特性，则取决于诸多因素，但通常都会包括以下特性：

① 名称

这是确定和区分不同模块的首要方式，也是称呼和调用模块的重要途径。通常情况下，模块名称会包含或暗示模块在系统中的位置、作用等相关信息。如，名称为 A_B_C 的模块

可能表示该模块是模块 B 的子模块，而模块 B 又是模块 A 的子模块。又如，名称为 system_markID 的模块可能表示该模块是系统中生成 ID 的模块。

② 责任和义务

模块责任和义务特性帮助识别模块在系统中的作用，并确定其脱离于名称的身份。尽管模块的名称可以提示读者有关该模块的作用，但是有关模块“责任和义务”的陈述，将可以更可靠地确定模块的作用。因此，应该在编档时对模块的“责任和义务”进行充分、详细、唯一的描述，以使读者能够方便、高效、无歧义地了解模块的作用。

③ 接口的可见性

通过在模块接口文档中确切的规定要求模块提供的操作，可以精确地确定模块在系统中的作用。通常，模块有且只有一个入口和一个出口，但却可以合法地拥有零个、一个或若干个接口。

模块接口的可见性会因模块所处位置不同，从不同的视角观察而有所不同。例如，在为具有“部分关系”编档的视图中，子模块的某些接口仅为内部用途而存在。换句话说，这些接口只用于封闭在父模块中的子模块。这样，这些接口在上下文之外就不可见了，因此，它们也就不和父接口发生直接的联系了。此时，对父模块而言，子模块的接口就是不可见的，如图 3.4 所示。

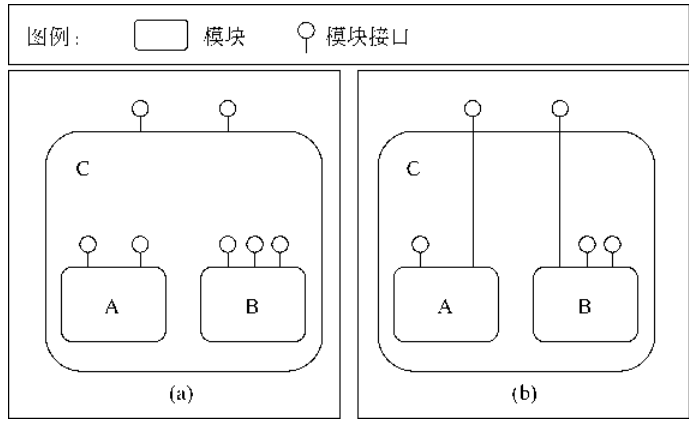

图 3.4　模块接口

对那些与父接口有直接联系的接口，则有不同的策略，如图 3.4(a)表示的是一个封装策略。在该策略下，子模块的接口被隐藏，仅有父模块可以使用子模块的接口，并映射其所有请求，而父模块之外的所有模块都将不能使用封闭子模块中的功能和接口。

另一方面，聚集模块的接口可以是该聚合的接口子集。换句话说，密闭模块只能聚集一个模块集，并有选择地解密自己的部分责任和义务。在软件系统中，层和子系统通常就是用这种方式定义的。如图 3.4(b)所示，模块 C 是一个密闭模块，由模块 A 和模块 B 聚合而成。从图中可以看到，模块 A 和模块 B 的接口集合中一个有选择解密的子集，现在是模块 C 的接口。

④ 实现信息

模块作为软件系统的实现单元，从构建包含模块的系统以及管理模块开发的需要出发，