软件工程导论

复习总结

猎奇人只爱无机盐
2023.6.25

目录

软件工程

第一章软件工程概论

第一节:软件和软件危机

一：软件的概念、特点与发展

（1）软件发展的三个阶段

程序设计阶段（50~60 年代）

程序系统阶段（60~70 年代）

软件工程阶段（70 年代以后）

（2）软件的概念

软件：是计算机系统中与硬件相互依存的另一部分，包括程序、数据及其相关文档的完整集合

数据：是使程序能够适当处理信息的数据结构

程序：是能够完成预定功能和性能的可执行指令序列

文档：是开发、使用和维护过程中程序所需要的图文资料

（3）软件的特点

① 软件本身具有复杂性

② 软件成本高昂

③ 软件未摆脱手工开发方式

④ 软件维护与硬件维护有本质区别，维护难度高

⑤ 软件开发不是传统的硬件制造过程

⑥ 软件是一种逻辑实体，无磨损性

二：软件危机

（1）软件危机的概念

软件危机：在计算机软件开发和维护过程中所遇到的一系列严重问题。主要包含两个方面

如何开发软件，以满足日益增长的软件需求

如何维护数量不断膨胀的已有软件

（2）软件危机的表现

①对软件开发成本和进度估算不准确

实际成本比估计成本有可能高出一个数量级,实际进度比预期进度拖延几个月甚至几年的现象并不罕见。这种现象降低了软件开发组织的信誉。而为了赶进度和节约成本所采取的一些权宜之计又往往损害了软件产品的质量，从而不可避免地会引起用户的不满用户对已完成软件不满意

②软件质量不可靠

软件可靠性和质量保证的确切的定量概念刚刚出现不久，软件质量保证技术(审查、复审、程序正确性证明和测试)还没有坚持不懈地应用到软件开发的全过程中，这些都导致软件产品发生质量问题。

③软件不可维护

很多程序中的错误是非常难改正的，实际上不可能使这些程序适应新的硬件环

境，也不能根据用户的需要在原有程序中增加一些新的功能。“可重用的软件”还是一个没有完全做到的、正在努力追求的目标，人们仍然在重复开发类似的或基本类似的软件

④没有适当的文档资料

⑤软件成本在计算机系统中所占比例逐年上升

由于微电子学技术的进步和生产自动化程度的不断提高，硬件成本逐年下降，然而软件开发需要大量人力，软件成本随着通货膨胀以及软件规模和数量的不断扩大而持续上升

⑥软件开发生产率低

软件产品“供不应求”的现象使人类不能充分利用现代计算机硬件提供的巨大潜力。

（3）软件危机产生的原因

A：主观原因

① 忽视需求分析

② 轻视软件维护

③ 没有认识到程序只是软件的一部分（很多人的共性问题）

④ 没有认识到软件开发只是软件漫长生命周期中一个比较次要的阶段

⑤ 越到后期如果引入变动则代价越高

B：客观原因

① 软件是逻辑实体，具有不可见性，所以管理和控制较为困难

② 软件不会磨损，维护意味着需要修改原来的设计，维护困难

③ 软件规模庞大，程序复杂性随规模增加而增加

（4）解决方法

①对计算机软件应该有正确的认识

应该彻底消除在计算机系统早期发展阶段形成的“软件就是程序”的错误观念。软件是程序、数据及相关文档的完整集合。其中，程序是能够完成预定功能和性能的可执行的指令序列;数据是使程序能够适当地处理信息的数据结构;文档是开发、使用和维护程序所需要的图文资料

②要吸取和借鉴人类长期从事各种工程项目积累的原理、概念、技术和方法

③积极开发和使用计算机辅助开发软件

④探索更好更有效的管理措施和手段对开发过程进行控制和管理

第二节：软件工程

一：软件工程

（1）软件工程的定义

软件工程：采用工程的概念、原理、技术和方法来开发与维护软件，把经过时间考验而证明正确的管理技术和当前能够得到的最好的技术方法结合起来，经济的开发出高质量的软件并维护它

（2）软件工程的本质特征

① 关注大型程序的构造

把一个人在较短时间内写出的程序称为小型程序，而把多人合作用时半年以上才写出的程序称为大型程序。传统的程序设计技术和工具是支持小型程序设计的，不能简单地把这些技术和工具用于开发大型程序。

② 中心课题是控制复杂性

软件所解决的问题通常十分复杂，不得不把问题分解，使得分解出的每个部分是可理解的，而且各部分之间保持简单的通信关系。用这种方法并不能降低问题的整体复杂性，但是却可使它变成可以管理的。

③软件经常变化

绝大多数软件都模拟了现实世界的某一部分。 现实世界在不断变化，软件为了不被很快淘汰，必须随着所模拟的现实世界一起变化。因此，在软件系统交付使用后仍然需要耗费成本，而且在开发过程中必须考虑软件将来可能发生的变化

④开发效率非常重要

社会对新应用系统的需求超过了人力资源所能提供的限度，软件供不应求的现象日益严重。软件工程的一个重要课题就是，寻求开发与维护软件的更好更有效的方法和工具

⑤开发人员和谐合作是关键

软件处理的问题十分庞大，必须多人协同工作才能解决这类问题。为了有效地合作，必须明确地规定每个人的责任和相互通信的方法，每个人还必须严格地按规定行事，纪律是成功地完成软件开发项目的一个关键

⑥软件需要有效支持用户

开发软件的目的是支持用户的工作。软件提供的功能应该能有效地协助用户完成他们的工作。有效地支持用户意味着必须仔细地研究用户，以确定适当的功能需求、可用性要求及其他质量要求。还意味着软件开发不仅应该提交软件产品，而且应该写出用户手册和培训材料

⑦软件开发者替代其他领域人员创造产品

软件工程师是软件设计、软件体系结构、测试或统一建 模语言等方面的专家，但他们不仅缺乏应用领域和文化领域的实际知识，还缺乏该领域的文化知识，这是软件开发项目出现问题的常见原因

（3）软件工程基本原理

①按软件生存期分阶段制定计划并认真实施

在软件开发与维护的漫长的生命周期中，需要完成许多性质各异的工作。应该把软件生命周期划分成若干个阶段，并相应地制定出切实可行的计划，然后严格按照计划对软件的开发与维护工作进行管理。不同层次的管理人员都必须严格按照计划各尽其职地管理软件开发与维护工作，绝不能受客户或上级人员的影响而擅自背离预定计划

②坚持进行阶段评审

由于大部分错误都是在编码之前造成的，并且错误发现与改正的越晚，所需付出的代价也越高,所以软件的质量保证工作不能等到编码阶段结束之后再进行。因此，在每个阶段都进行严格的评审，以便尽早发现在软件开发过程中所犯的错误，是一条必须遵循的重要原则

③坚持严格的产品控制

当改变需求时，为了保持软件各个配置成分的一致性， 必须实行严格的产品控制，其中主要是实行基准配置管理。所谓基准配置又称为基线配置，它们是经过阶段评审后的软件配置成分

④使用现代程序设计技术

采用先进的技术不仅可以提高软件开发和维护的效率，而且可以提高软件产品的质量

⑤结果能够得到清楚的审查

软件产品不同于一般的物理产品，它是看不见摸不着的逻辑产品。软件开发人员的工作进展情况可见性差,难以准确度量。为了提高软件开发过程的可见性，更好地进行管理，应该根据软件开发项目的总目标及完成期限,规定开发组织的责任和产品标准，从而使得所得到的结果能够清楚地审查

⑥用人少儿精

软件开发小组的组成人员的素质应该好，而人数则不宜过多。素质高的人员的开发效率比素质低的人员的开发效率可能高几倍至几十倍，而且素质高的人员所开发的软件中的错误明显少于素质低的人员所开发的软件中的错误

⑦承认不断改进软件工程实践的必要性

为了保证软件开发与维护的过程能赶上时代前进的步伐，能跟上技术的不断进步，不仅要积极主动地采纳新的软件技术，而且要注意不断总结经验。例如，收集进度和资源耗费数据，收集出错类型和问题报告数据等

二：软件工程方法学

（1）概念

软件工程方法学：把在软件生命周期全过程中使用的一整套技术方法的集合称之为方法学，也称为泛型。软件工程方法学包含三个要素：方法、工具、过程

方法：完成软件开发各项任务的技术方法，回答“怎么做”的问题

工具：为运用方法提供的自动或半自动软件工程支撑环境

过程：是为了获得高质量软件所需要完成的一系列任务框架，回答“何时做”的问题

（2）分类

传统方法学（生命周期方法学）

① 采用结构化技术完成软件开发各项任务
② 把软件生命周期的全过程依次划分为若干阶段
③ 每个阶段开始和结束都有严格标准
④ 每个阶段结束后要有严格审查

面向对象方法学

① 把对象作为融合了数据及在数据上的操作行为的统一的软件构件
② 把所有对象划分为类
③ 按照父类与子类的关系，把若干类组成层次结构的系统
④ 对象彼此间仅能通过发送消息互相联系

第三节：软件生命周期

一：软件定义

（1）问题定义

问题定义：弄清用户要解决什么问题

通过对客户的访问调查，系统分析员扼要地写出关于 问题性质、工程目标和工程规模的书面报告，经过讨论和必要的修改之后这份报告应该得到客户的确认

（2）可行性研究

可行性研究：确定问题是否可行

为了回答这个问题，系统分析员需要进行一次大大压缩和简化了的系统分析和设计过程。可行性研究阶段的任务是研究问题的范围，探索这个问题是否值得去解，是否有可行的解决办法。可行性研究的结果是客户作出是否继续进行这项工程的决定的重要依据，只有投资可能取得较大效益的那些工程项目才值得继续进行下去。

（3）需求分析

需求分析：为了解决这个问题，系统需要具备怎样的功能

系统分析员必须和用户密切配合，充分交流信息，以得出经过用户确认的系统逻辑模型。用数据流图、数据字典和简要的算法表示系统的逻辑模型。这个阶段的一项重 要任务，是用正式文档准确地记录对目标系统的需求，这份文档通常称为软件需求规格说明书(SRS)。

二：软件开发

（1）总体设计

总体设计：设计软件结构，确定程序由哪些模块组成以及模块间的关系

软件工程师应该用适当的表达工具描述每种方案,分析每种方案,推荐一个最佳方案并制定出详细计划。另一项主要任务就是设计程序的体系结构，即确定程序由哪些模块组成以及模块间的关系。

（2）详细设计

详细设计：针对每个模块，设计详细规格说明，确定算法和数据结构

详细设计阶段的任务就是把设计方案具体化，也就是回答:“应该怎样具体地实现这个系统呢?”这个阶段的任务是设计出程序的详细规格说明。在这个阶段将详细地设计每个模块，确定实现模块功能所需要的具体算法和数据结构

（3）编码和单元测试

编码和单元测试：将详细设计内容用语言实现，并测试每个模块

编码和单元测试阶段的关键任务是写出正确的、容易理解的、容易维护的程序模块。程序员把详细设计的结果翻译成用选定的高级编程语言书写的程序，编写出的每-一个模块， 并对编写好的各个模块进行测试

（4）综合测试

综合测试：通过各种类型测试使软件达到预定要求

最基本的测试是集成测试和验收测试

三：软件维护

①改正性维护，即诊断和改正在使用过程中发现的软件错误;

②适应性维护，即修改软件以适应环境的变化;

③完善性维护，即根据用户的要求改进或扩充软件使它更完善;

④预防性维护，即修改软件，为将来的维护活动预先做准备。

第四节：软件过程及相关模

一：软件过程概念

（1）定义

软件过程：是为了获得高质量软件所需要完成的一系列任务框架，它规定了完成任务的工作步骤。通常用软件生命周期模型来描述软件过程。常见模型有：

① 瀑布模型
② 快速原型模型
③ 增量模型
④ 螺旋模型
⑤ 喷泉模型
⑥ 其他模型

（2）构成

科学、有效的软件过程应该定义一-组适合于所承担的项目特点的任务集合。一个

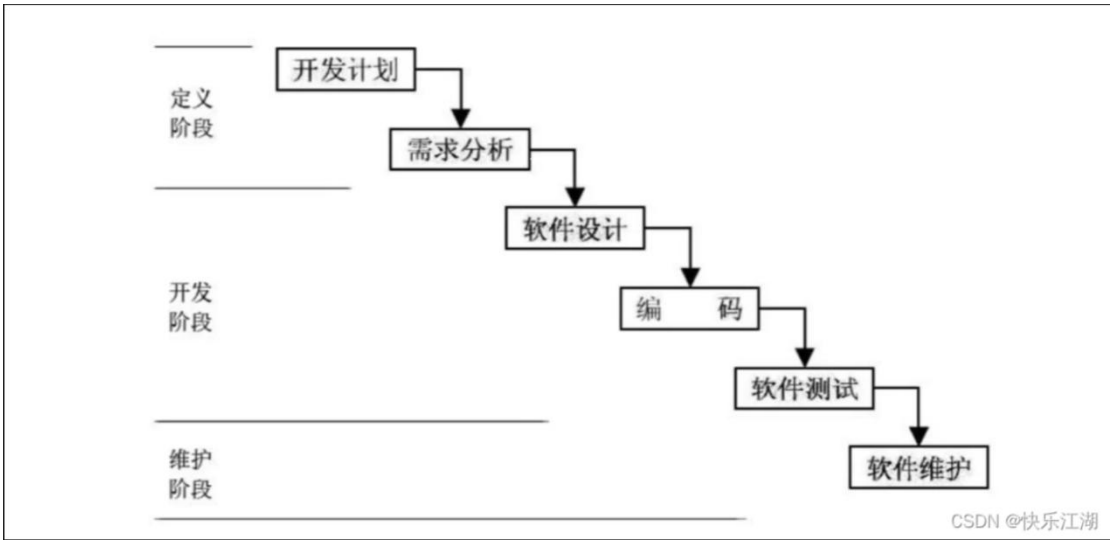
任务集合包括一组软件工程任务、里程碑和应该交付的产品。

二：主要模型

（1）瀑布模型

A：定义

瀑布模型：将软件生命周期的各项活动规定为依照固定顺序连接的若干阶段工作，最终得到软件产品



B：特点

①阶段间具有顺序性和依懒性

- 必须等前一阶段的工作完成之后，才能开始后一阶段的工作;
- 前一阶段的输出文档是后一阶段的输入文档
- 只有前一阶段的输出文档正确，后一阶段的工作才能获得正确的结果

②推迟实现的观点

瀑布模型在编码之前设置了系统分析与系统设计的各个阶段，分析与设计阶段的基本任务规定，在这两个阶段主要考虑目标系统的逻辑模型，不涉及软件的物理实现。

清楚地区分逻辑设计与物理设计，尽可能推迟程序的物理实现，是按照瀑布模型开发软件的一-条重要的指导思想。

③质量保证的观点

为了保证所开发的软件的质量,在瀑布模型的每个阶段都必须完成规定的文档，没有交出合格的文档就是没有完成该阶段的任务。

完整、准确的合格文档不仅是软件开发时期各类人员之间相互通信的媒介，也是运行时期对软件进行维护的重要依据。

越是早期阶段犯下的错误，暴露出来的时间就越晚，排除故障改正错误所需付出的代价也越高。

因此，及时审查，是保证软件质量、降低软件成本的重要措施。每个阶段结束前都要对所完成的文档进行评审，以便尽早发现问题，改正错误。

C：优缺点

优点：

① 强迫开发人员使用规范的方法

② 严格规定了每个阶段提交的文档

③ 要求每个阶段交出的所有产品都必须经过质量保证小组的仔细验证

④ 对文档的约束，使软件维护变得容易一些，且能降低软件预算

缺点：

① 在软件开发的初期阶段就要求作出正确的、全面的、完整的需求分析对许多应用软件来说是极其困难的

② 在需求分析阶段，当需求确定后，无法及时验证需求是否正确、完整

③ 作为整体开发的瀑布模型，由于不支持产品的演化，缺乏灵活性，对开发过程中很难发现的错误，只有在最终产品运行时才能暴露出来，从而使软件产品难以维护

D:：适用范围

① 用户的需求非常清楚全面，且在开发过程中没有或很少变化;

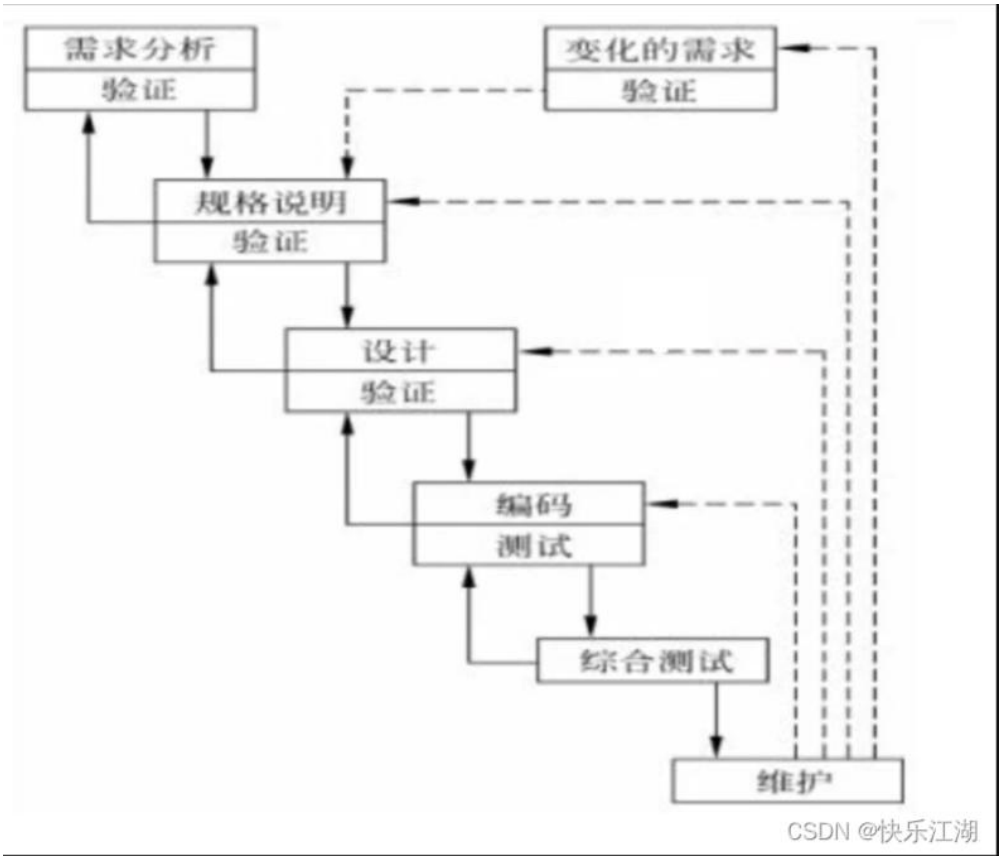
② 开发人员对软件的应用领域很熟悉;

③ 用户的使用环境非常稳定;

④ 开发工作对用户参与的要求很低。.

E：实际的瀑布模型

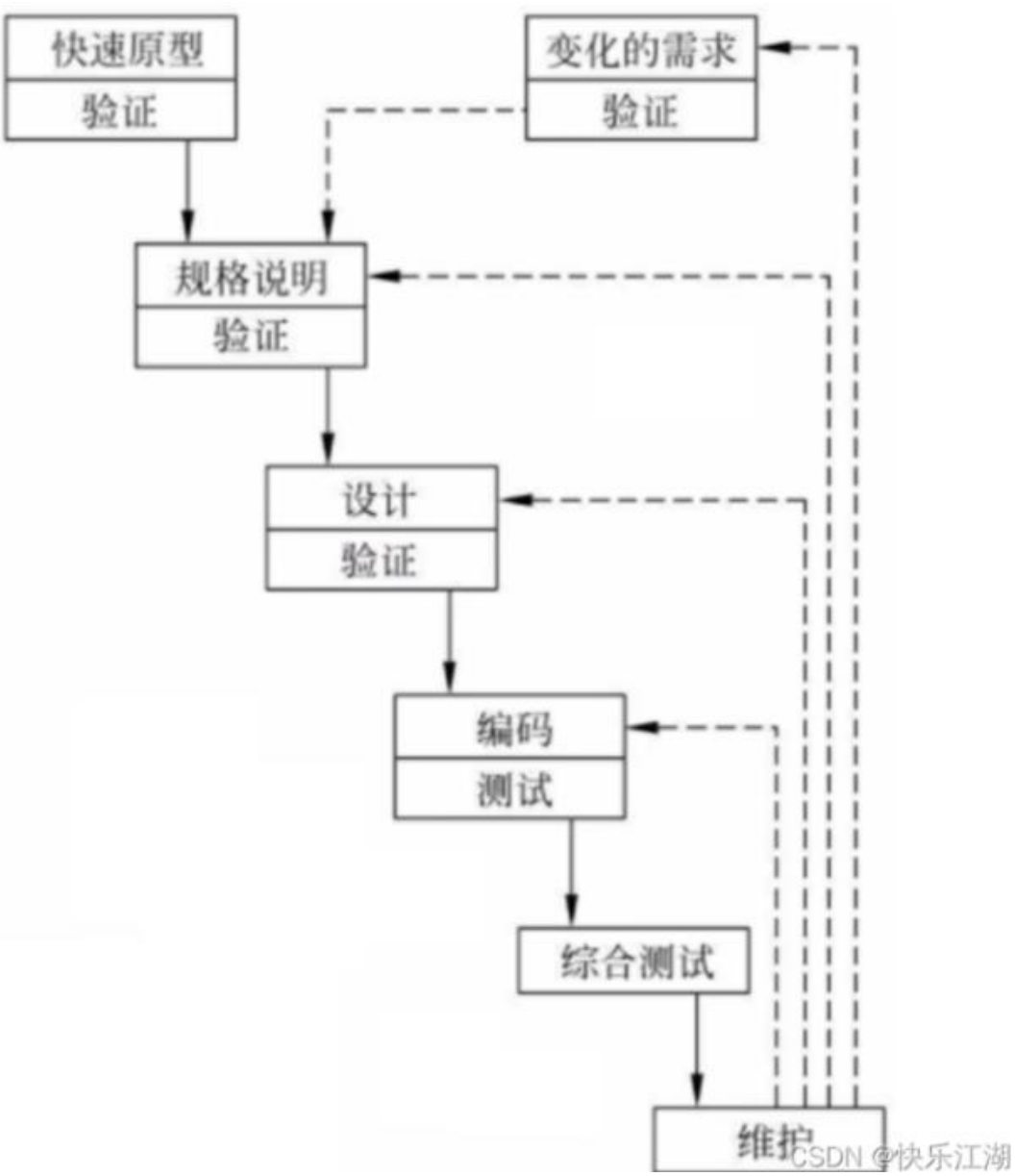
在设计阶段可能发现规格说明文档中的错误，而设计上的缺陷或错误可能在实现过程中显现出来，在综合测试阶段将发现需求分析、设计或编码阶段的许多错误。因此，实际的瀑布模型是带“反馈环”的，如下图所示(图中实线箭头表示开发过程，虚线箭头表示维护过程)。当在后面阶段发现前面阶段的错误时，需要沿图中左侧的反馈线返回前面的阶段，修正前面阶段的产品之后再回来继续完成后面阶段的任务。



（2）快速原型模型

A：定义

快速原型模型：快速建立可运行的程序，它完成的功能往往是最终产品功能的一个子集



B：原理

快速原型模型的第一步是快速建立一个能反映用户主要需求的原型系统,让用户通过实践来了解目标系统的概貌。通常，用户试用原型系统之后会提出许多修改意见，开发人员按照用户的意见快速地修改原型系统，然后再次请用户试用,一旦用户认为这个原型系统确实能做他们所需要的工作,开发人员便可据此书写规格说明文档,根据这份文档开发出的软件便可以满足用户的真实需求。

C：优缺点

优点：

① 开发的软件产品通常满足用户需求

② 软件产品开发基本是线性过程

缺点：

① 准确原型设计困难

② 原型理解可能不同

③ 不利于开发人员创新

D：适用范围

① 对所开发的领域比较熟悉而且有快速的原型开发工具;

② 项目招投标时，可以以原型模型作为软件的开发模型;

③ 进行产品移植或升级时，或对已有产品原型进行客户化工作时。

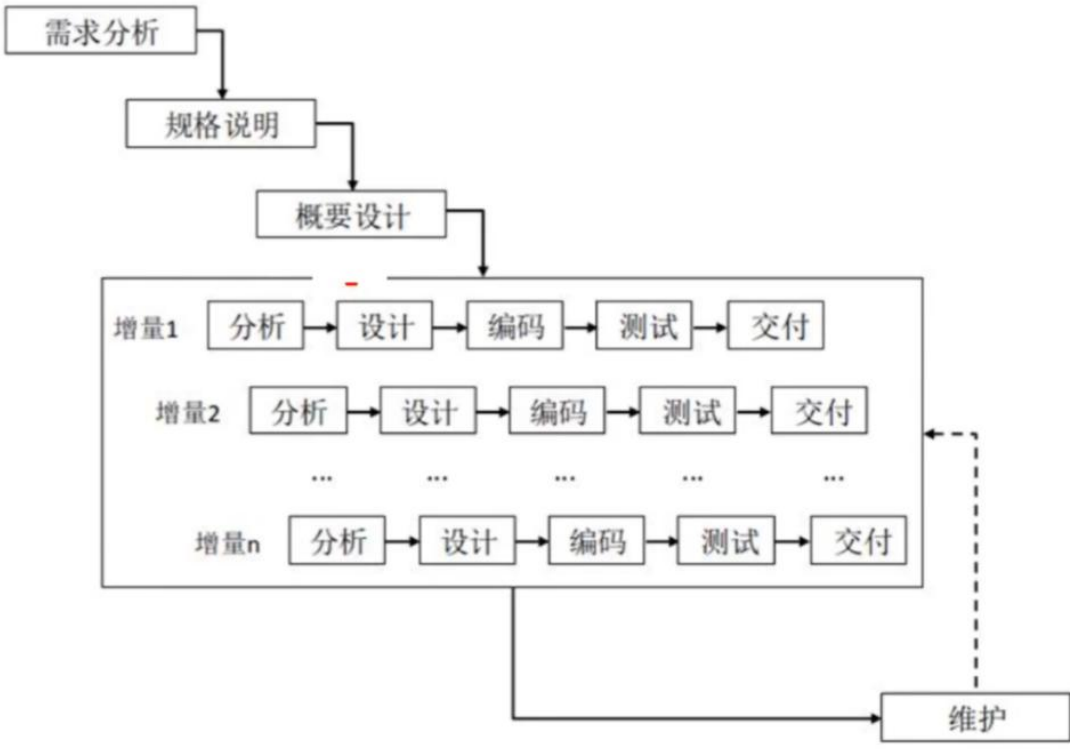
（3）增量模型（了解）

A：定义

增量模型：先完成一个系统子集的开发，再按同样的开发步骤增加功能，如此递增下去直至满足全部系统需求

每个构件由多个相互作用的模块构成，并且能够完成特定的功能。使用增量模型时，

第一个增量构件往往实现软件的基本需求，提供最核心的功能。把软件产品分解成增量构件时，应该使构件的规模适中。分解时唯一必须遵守的约束条件是:当把新构件集成到现有软件中时，所形成的产品必须是可测试的



B：优缺点

优点：

① 短时间内可提交完成部分功能

② 逐渐增加产品功能，用户适应产品快

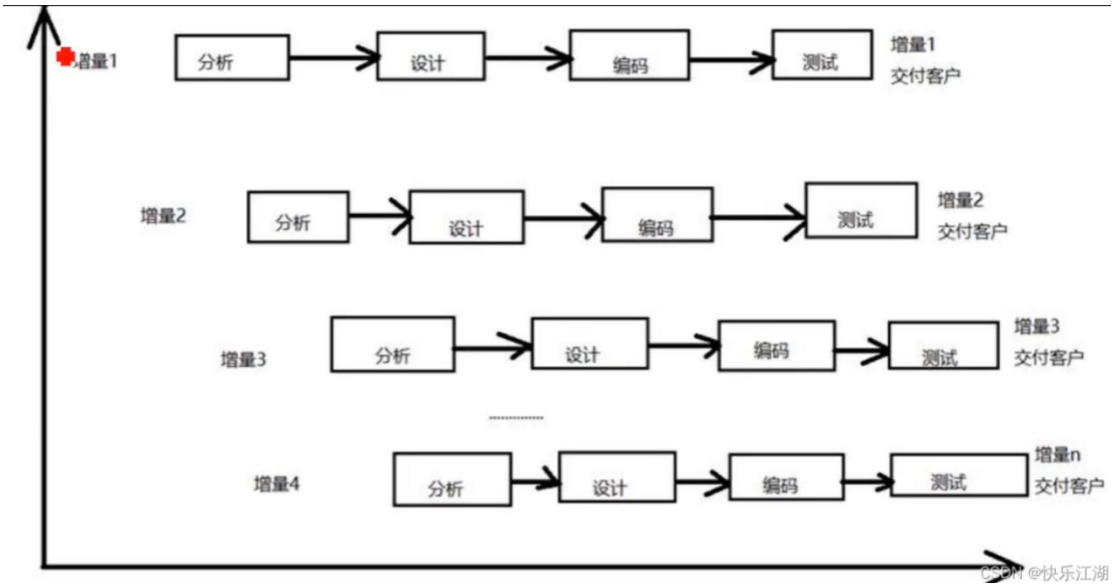
缺点：

① 增量构件划分以及集成困难

② 容易退化为边做边改模型

C：风险更大的增量模型

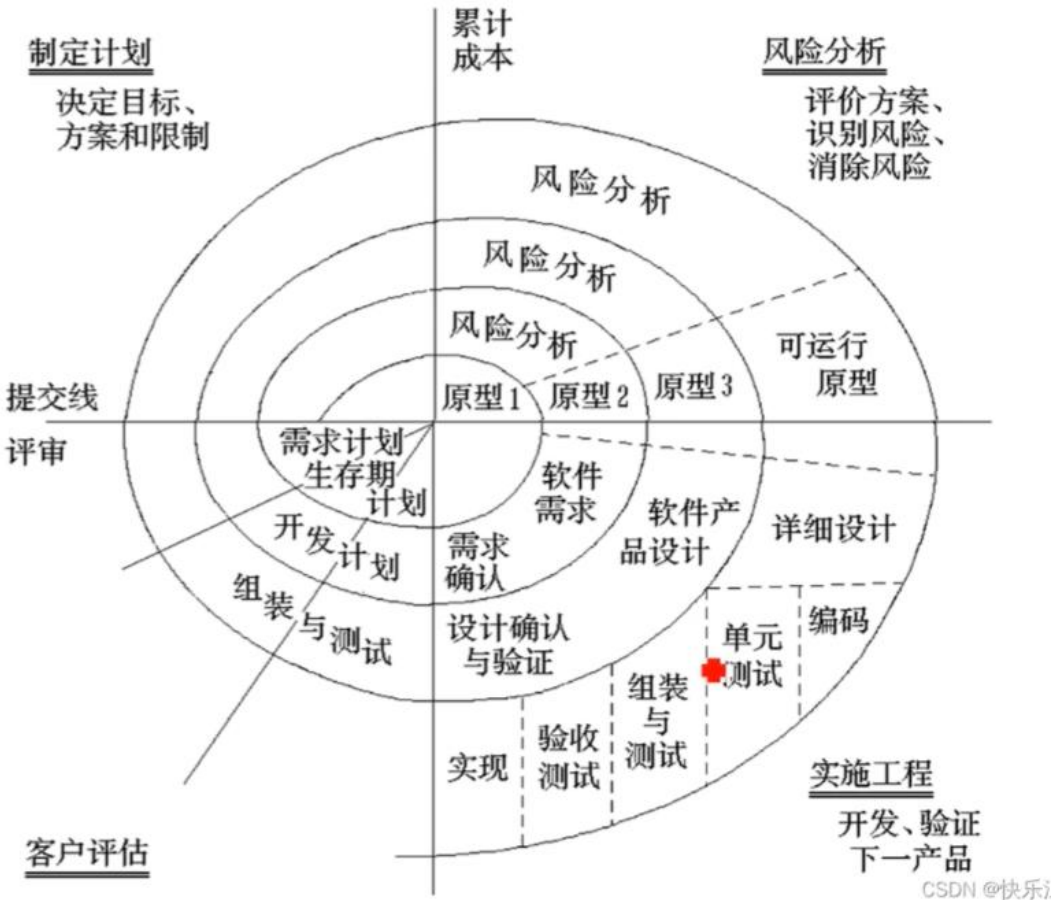
下图是一种风险更大的增量模型。用这种方式开发软件，不同的构件将并行地构建，因此有可能加快工程进度。但是，这种方法将有构件无法集成到一起的风险，除非密切地监控整个开发过程，否则整个工程可能毁于一旦。



（4）螺旋模型（了解）

A：定义

螺旋模型：在每个阶段之前都增加了风险分析过程的快速原型模型



B：优缺点

优点：

① 利于把软件质量作为软件的开发目标

② 减少测试

③ 维护和开发不分开

缺点：

风险估计困难

（5）喷泉模型

喷泉模型：典型的面向对象软件过程模型。体现了迭代和无缝的特性

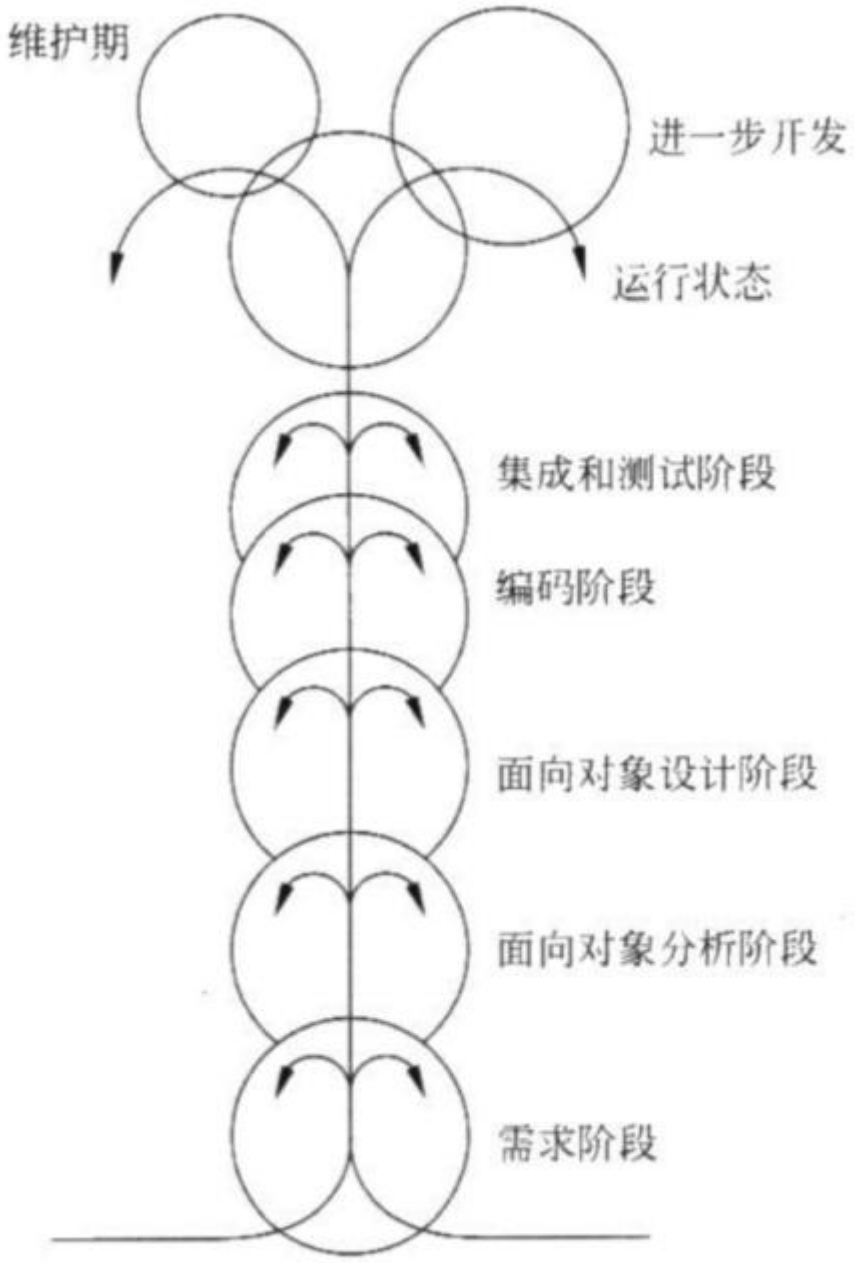


图 1-8 喷泉模型

CSDN @书

（6）其他模型

① Rational 统一过程

优点：

提高了团队生产力，在迭代的开发过程、需求管理、基于组建的体系结构、可视化软件建模、验证软件质量及控制软件变更等方面、针对所有关键的开发活动为每个开发成员提供了必要的准则、模版和工具指导，并

确保全体成员共享相同的知识基础。它建立了简洁和清晰的过程结构，为开发过程提供较大的通用性

缺点:

RUP 只是一个开发过程，并没有涵盖软件过程的全部内容，例如它缺少关于软件运行和支持等方面的内容，此外，他没有支持多项目的开发结构，这在一定程度上降低了在开发组织内大范围实现重用的可能性

适用范围：

大型的需求不断变化的复杂软件系统项目

② 敏捷过程与极限编程

适用范围：

适用于商业竞争环境下对小型项目提出的有限资源和有限开发时间的约束

③ 微软过程

适用范围：

适用于商业环境下具有有限资源和有限开发时间约束的项目的软件过程模式

第二章：可行性研究

第一节：可行性研究基本介绍

一：可行性研究的目的

可行性研究的目的：用最小的代价在最小的时间内确定问题是否可以被解决

并非任何问题都有简单明显的解决办法，事实上，许多问题不可能在预定的系统规模或时间期限之内解决。如果问题没有可行的解，那么花费在这项工程上的任何时间、人力、软硬件资源和经费，都是无谓的浪费

二：可行性研究的本质

可行性研究的本质：系统分析和设计过程的大大压缩和简化，在较高层次上以较为抽象的方式进行系统的分析和设计过程

三：可行性研究的任务

（1）最根本任务

可行性研究的最根本任务：对以后的行动方针提出建议

（2）具体任务

1.分析和澄清问题的定义

2.导出系统的逻辑模型

数据流图+数据字典

3.根据逻辑模型探索若干种可供选择的解法

4.研究每种解法的可行性

① 经济可行性：经济效益是否大于开发成本

② 技术可行性：现有技术能够实现

③ 操作可行性：系统操作方式是否可行

④ 其它可行性：法律、社会效益

二：可行性研究过程（步骤）

1：复查系统规模和目标

对问题定义阶段初步确定的规模和目标进行肯定或改正，并列出对目标系统的约束和限制

2：研究目前正在使用的系统

了解现有系统能够做什么，而不花费过多时间分析怎么实现这些功能

3：导出新系统的高层逻辑模型

现有物理系统->现有逻辑模型->目标逻辑模型->目标物理模型

4：进一步定义问题

分析员和用户一起再次复查系统

（步骤 1~4 构成一个循环）

5：导出和评价供选择的解法

① 技术角度排除不可行方案

② 操作可行性角度排除用户不能接受方案

③ 经济可行角度估算成本和收益

6：推荐行动方针

给出是否继续的结论

7：草拟开发计划

① 制定进度表

② 开发人员、计算机资源分析

③ 估计每阶段成本、下阶段详细分析

8：书写文档提交审查

三、可行性研究的必要性：

开发一个软件时，需要判断原定的系统模型和目标是否现实，系统完成后所能带来的效益是否大到值得投资开发这个系统的程度，如果做不到这些，那么花费在这些工程上的任何时间、人力、软硬件资源和经费，都是无谓的浪费。可行性研究的实质是要进行一次大大压缩简化了 的系统分析和设计过程，就是在较高层次上以较抽象的方式进行的系统分析和设计的过程。可行性研究的目的就是用最小的代价在尽可能短的时间内确定问题是否能够解决

涉及方面

经济可行性：经济效益是否大于开发成本

技术可行性：现有技术能够实现

操作可行性：系统操作方式是否可行

其它可行性：法律、社会效益

第三节：数据流图

一：数据流图的定义（DFD）

数据流图：描述信息流和数据从输入到输出所经受的变换。没有任何具体物理部件，只是描绘数据在软件中流动和被处理的逻辑过程

二：数据流图的特点

① 数据流图中没有具体的物理部件，只是描绘数据在软件中流动和被处理的逻辑过程

② 数据流图是系统逻辑功能的图形表示，是分析员与用户之间极好的通信工具

③ 设计时只需考虑系统必须完成的基本逻辑功能，不考虑怎样具体地实现这些功能

三：数据流图的符号

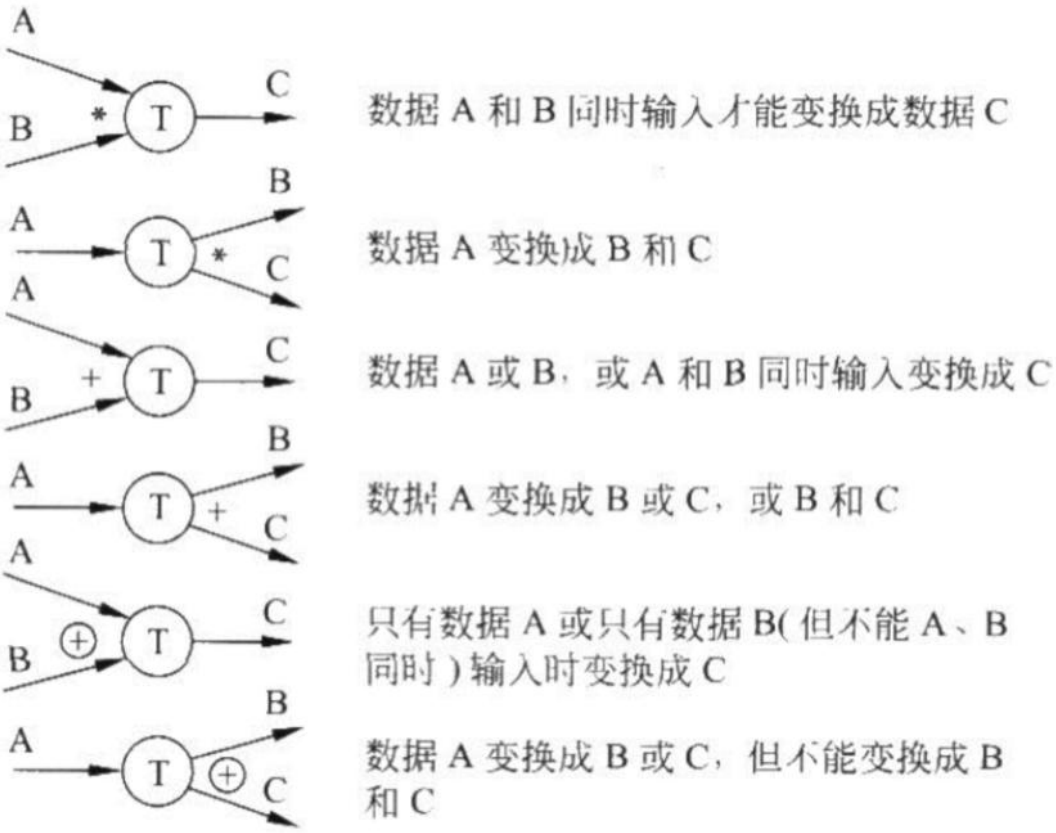
（1）符号

基本符号

源点/终点　变换数据的处理　数据存储　数据流

CSDN @快乐江湖

关于“数据流”的扩充



（2）注意（了解）

1.在数据流图中应该描绘所有可能的数据流向，而不应该描绘出现某个数据流的条件。

2.一个处理框可以代表一系列程序、单个程序或者程序的一个模块。

3.一个数据存储可以表示一个文件、文件的一部分、数据库的元素或记录的一部分等 。

4.数据存储是处于静止状态的数据，数据流是处于运动中的数据。

5.通常在数据流图中忽略出错处理。

6.表示数据的源点和终点相同的方法是再重复画一个同样的符号表示数据的终点。

7.代表同一事物的符号出现在 n 个地方，在这个符号的角上画(n-1) 条短斜线做标记。

四：数据流图示例

（1）示例 1

假设一家工厂的采购部每天需要一张订货报表，报表按零件编号排序，表中列出所有需要再次订货的零件。对于每个需要再次订货的零件应该列出下述数据:零件编号，零件名称，订货数量，目前价格，主要供应者，次要供应者。零件入库或出库称为事务，通过放在仓库中的 CRT 终端把事务报告给订货系统。当某种零件的库存数量少于库存量临界值时

就应该再次订货。画出上述订货系统的数据流图

A：首先从题目中提取四种成分

1：考虑数据的源点和终点

数据源点：仓库管理员

数据终点：采购员

2：考虑有哪些处理

“采购部需要报表”，所以需要一个产生报表的处理

仓库中的零件数量会发生改变，所以对事物进行的加工是另一个处理

3：考虑数据流

系统会把订货报表送给采购部，所以订货报表是一个数据流

事物需要从仓库送到系统中，所以事物是一个数据流

4：考虑数据存储

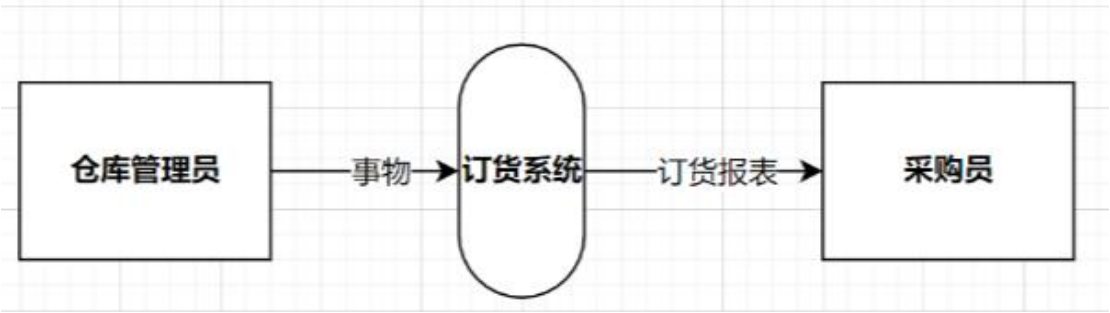
每当有一个事物发生时就应该立即处理，但是由于每天只产生一次订货报表。因此用于产生订货报表的数据必须存放一段时间，所以有一个数据存储

B：画数据流图

1：基本系统模型

任何系统的基本模型都由若干数据源点/终点以及一个处理组成，该处理代表系统对数据加工变换的基本功能

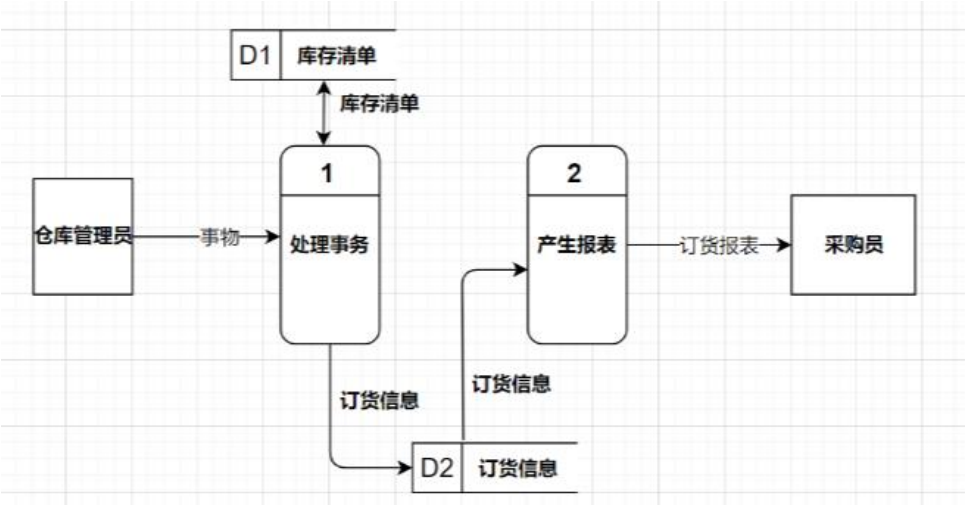
如下：



2：细化模型，描绘系统主要功能

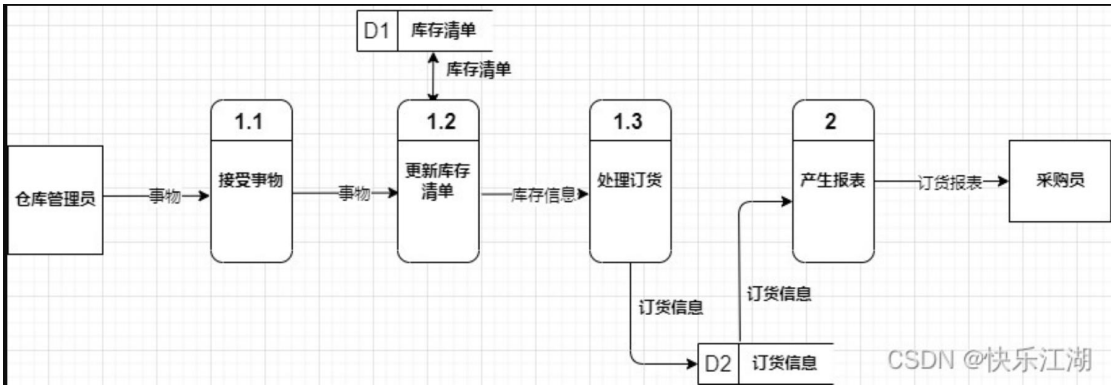
产生报表和处理事物是系统必须完成的两个主要功能

细化后增加了两个数据存储：处理事物需要库存清单数据；产生报表和处理事物需要订货信息



3：进一步细化功能级数据流图中描绘的系统主要功能

当一个事物发生使必须先接受它，随后按照事物的内容修改库存清单，最后如果更新后的库存量少于临界值，需要再次订货



4：考虑是否继续分解和细化

一旦在想要分解时产生了诸如“如何具体地实现一个功能”这样的疑问时就表明不需要分解了

5：检查

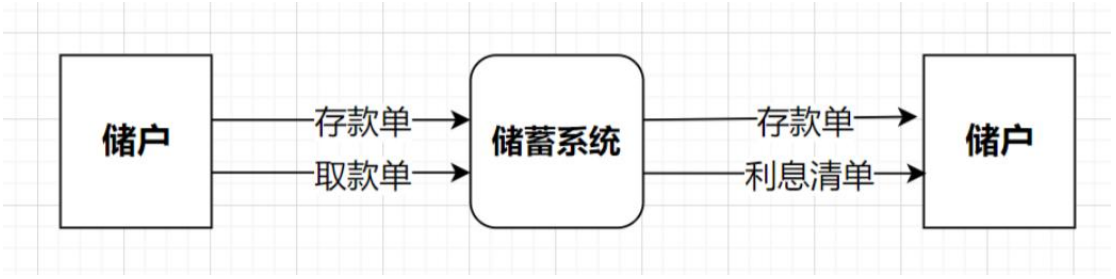
分层必须保证信息的连续性

注意编号的处理

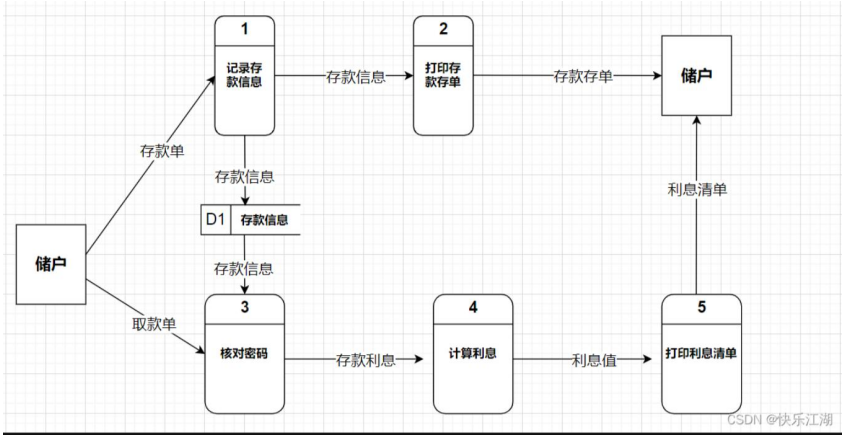
（2）示例 2

银行计算机储蓄系统的工作过程大致如下：储户填写的存款单或取款单由业务员键入系统，如果是存款则系统记录存款人姓名、住址(或电话号码)、身份证号码、存款类型、存款日期、到期日期、利率及密码(可选)等信息，并印出存单给储户；如果是取款而且存款时留有密码，则系统首先核对储户密码，若密码正确或存款时未留密码，则系统计算利息并印出利息清单给储户

第一层：



第二层：



五：数据流图的用途

（1）目的

1.作为交流信息得工具

2.作为分析和设计的工具

（2）自动化边界

以图中不同处理的定时要求为指南,能够在数据流图.上画出许多组自动化边界,每组自动化边界可能意味着一个不同的物理系统，因此可以根据系统的逻辑模型考虑系统的物理实现

第四节：数据字典

一：相关概念

（1）定义

数据字典：是关于数据的信息集合，即对数据流图中包含的所有元素定义的集合

（2）作用

在软件分析和设计的过程中给人提供关于数据的描述信息

（3）意义（了解）

① 数据流图和数据字典共同构成系统的逻辑模型

② 没有数据字典，数据流图就不严格，然而没有数据流图，数据字典也难以发挥作用

③ 只有数据流图和对数据流图中每个元素的精确定义放在一起，才能共同构成系统的规格说明

二：内容

一般来说，数据字典应该由对下列 4 类元素的定义组成

① 数据流

② 数据元素

③ 数据存储

④ 处理

三：定义数据的方法

（1）数据元素组成数据的方式

① 顺序

② 选择

③ 重复

④ 可选

（2）符号表示

=	等价于
+	和（即连接两个分量）
[]	或（即从括号内的分量中选择一个）
|	隔开供选择的分量
{}	重复
()	可选（即括号内的分类可有可无）

四：数据字典示例

北京某高校可用的电话号码有以下几类:校内电话号码由 4 位数字组成，第一位数字不是 0。校外电话又分为本市电话和外地电话两类。拨校外电话需要先拨 0，若是本市电话则接着拨 8 位数字(第一位不是 0)，若是外地电话则拨 3 位区码后再拨 8 位电话号码(第一位

不是 0)

电话号码=[校内电话 | 校外电话]

校内电话=非零数字+三位数字

非零数字=[1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9]

三位数字=3{数字}3

数字=[0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9]

校外电话=[本市电话 | 外地电话]

本市电话=0 + 八位非零开头数字

八位非零开头数字=非零数字+七位数字

七位数字=7{数字}7

外地电话=0 + 三位区码 + 八位非零开头数字

三位区码=三位数字

五：用途

1.作为分析阶段的工具

在数据字典中建立的一组严密一致的定义很有助于改进分析员和用户之间的通信，可以消除许多可能的误解。对数据的这一系列严密一致的定义有助于改进在不同的开发人员或不同的开发小组之间的通信。如果要求所有开发人员都根据公共的数据字典描述数据和设计模块，则能避免许多麻烦的接口问题

2.数据字典中包含的数据元素的控制信息是很有价值的

数据字典列出了使用一个给定的数据元素的所有程序(或模块)，所以很容易估计改变一个数据将产生的影响，并且能对所有受影响的程序或模块做出相应的改变。

3. 数据字典是开发数据库的第一步，而且是很有价值的一步

第三章：需求分析

第一节：需求分析相关概念

一：需求分析相关概念

（1）定义

需求分析是软件定义时期的最后一个阶段,它的基本任务是准确地回答“系统必须做什么”这个问题，目标系统提出完整、准确、清晰、具体的要求。在需求分析阶段结束之前，系统分析员应该写出软件需求规明书，以书面形式准确地描述软件需求

（2）必要性（了解）

为了开发出真正满足用户需求的软件产品，首先必须知道用户的需求。对软件需求的深入理解是软件开发工作获得成功的前提条件，不论人们把设计和编码工作做得如何出色，不能真正满足用户需求的程序只会令用户失望，给开发者带来烦恼

（3）准则

① 必须理解并描述问题的信息域，根据这条准则应该建立数据模型

② 必须定义软件应完成的功能，这条准则要求建立功能模型

③ 必须描述作为外部事件结果的软件行为，这条准则要求建立行为模型

④ 必须对描述信息、功能和行为的模型进行分解，用层次的方式展示细节

二：需求分析的任务

（1）确定对系统的综合要求

① 功能要求：系统必须提供的服务功能

② 性能要求：系统必须满足的约束条件（如响应速度、安全性等）

③ 可靠性和可用性需求：可靠性定量、可用性量化

④ 出错处理需求： 错误响应机制，说明系统对环境错误应该如何响应

⑤ 接口需求：

1) 用户接口需求
2) 硬件接口需求
3) 软件接口需求
4) 通信接口需求

⑥ 约束： 用户或环境强加的限制条件（如工具、语言等）

⑦ 逆向需求： 系统不应该做什么

⑧ 将来可能提出要求： 将来可能需要实现的需求

（2）分析系统的数据要求

A：意义（了解）

任何一个软件系统本质上都是信息处理系统,系统必须处理的信息和系统应该产生的信息在很大程度上决定了系统的面貌。因此，必须分析系统的数据要求，这是软件需求分析的一个重要任务

B：工具

常用的方法：

建立数据模型

常用图形工具：

层次方框图

warnier 图

（3）导出系统的逻辑模型

综合分析结果可以导出系统的详细的逻辑模型，通常用数据流图、实体联系图、状态转换图、数据字典和主要的处理算法描述这个逻辑模型。

（4）修正系统开发计划

根据在分析过程中获得的对系统的更深入更具体的了解，可以比较准确地估计系统的成本和进度，修正以前制定的开发计划。

三：与用户沟通获取需求的方法（了解）

（1）访谈

A：基本形式

正式会谈：系统分析员将提出一-些事先准备好的具体问题。

非正式会谈：分析员将提出一些用户可以自由回答的开放性问题，鼓励被访问人员说出自己的想法

B：技术方法

调查表技术：当需要调查大量人员的意见时，向被调查人分发调查表是-一个十分有效的做法。经过仔细考虑写出的书面回答可能比被访者对问题的口头回答更准确。分析员仔细阅读收回的调查表，然后再有针对性地访问一些用户，以便向他们询问在分析调查表时发现的新问题

情景分析技术：是对用户将来使用目标系统解决某个具体问题的方法和结果进行分析。系统分析员利用情景分析技术，往往能够获知用户的具体需求。

（2）面向数据流自顶向下求精

结构化分析方法是面向数据流自顶向下逐步求精进行需求分析的方法。通过可行性研究已经得出了目标系统的高层数据流图，需求分析的目标之一就是把数据流和数据存储定义到元素级

（3）简易的应用规格说明技术

A：定义

简易的应用规格说明技术是一种面向团队的需求收集法。这种方法提倡用户与开发者密切合作，共同标识问题，提出解决方案要素，商讨不同方案并指定基本需求。是信息系统领域使用的主流技术

B：应用过程

① 进行初步的访谈并确定会议方案

② 进行会议准备

③ 开会讨论

④ 会后总结并起草规格说明书

（4）快速建立软件原型

A：定义

快速原型是快速建立起来的旨在演示目标系统主要功能的可运行的程序。构建原型的要点是，它应该实现用户看得见的功能，省略目标系统的“隐含”功能。快速建立软件原型是最准确、最有效、最强大的的需求分析技术

B：特性

快速

容易修改

四：分析建模与规格说明（了解）

（1）模型

是指为了理解事物而对事物做出的一种抽象，是对事物的一种无歧义的书面描述

（2）模型分类

数据模型（实体-联系图）：描绘数据对象及数据对象之间的关系

功能模型（数据流图）：描绘数据在系统中流动时被处理的逻辑过程，指明系统具有的变换数据的功能

行为模型（状态转换图）：描绘系统的各种行为模式在不同状态间转换的方式

第二节：实体联系图（ER 图）

一：数据模型

（1）定义

为了把用户的数据要求清楚、准确地描述出来，通常建立一个概念性的数据模型(信息模型)。概念性数据模型是一种面向问题的数据模型，是按照用户的观点对数据建立的模型。它描述了从用户角度看到的数据，它反映了用户的现实环境，而且与在软件系统中的实现方法无关

（2）构成

数据对象：是对软件必须理解的复合信息的抽象

复合信息是指具有一系列不同性质或属性的事物，仅有单个值的事物不是数据对象

属性：属性定义了数据对象的性质。必须把一个或多个属性定义为“标识符”，即当希望找到数据对象的一个实例时，用标识符属性作为“关键字”(“键”)。应该根据对所要解决的问题的理解，来确定特定数据对象一组合适的属性

联系：数据对象彼此之间相互连接的方式称为联系，也称为关系。联系也可能有属性。联系可分为以下 3 种类型

一对一（1:1）

一对多（1:N）

多对多（M:N）

二：实体联系图（E-R 图）

（1）定义

使用实体联系图可以建立数据模型，利用 E-R 图描绘的数据模型称之为 E-R 模型

实体： 描述的数据对象

属性： 描述数据对象的性质

联系： 描述数据对象之间的交互方式

（2）符号表示

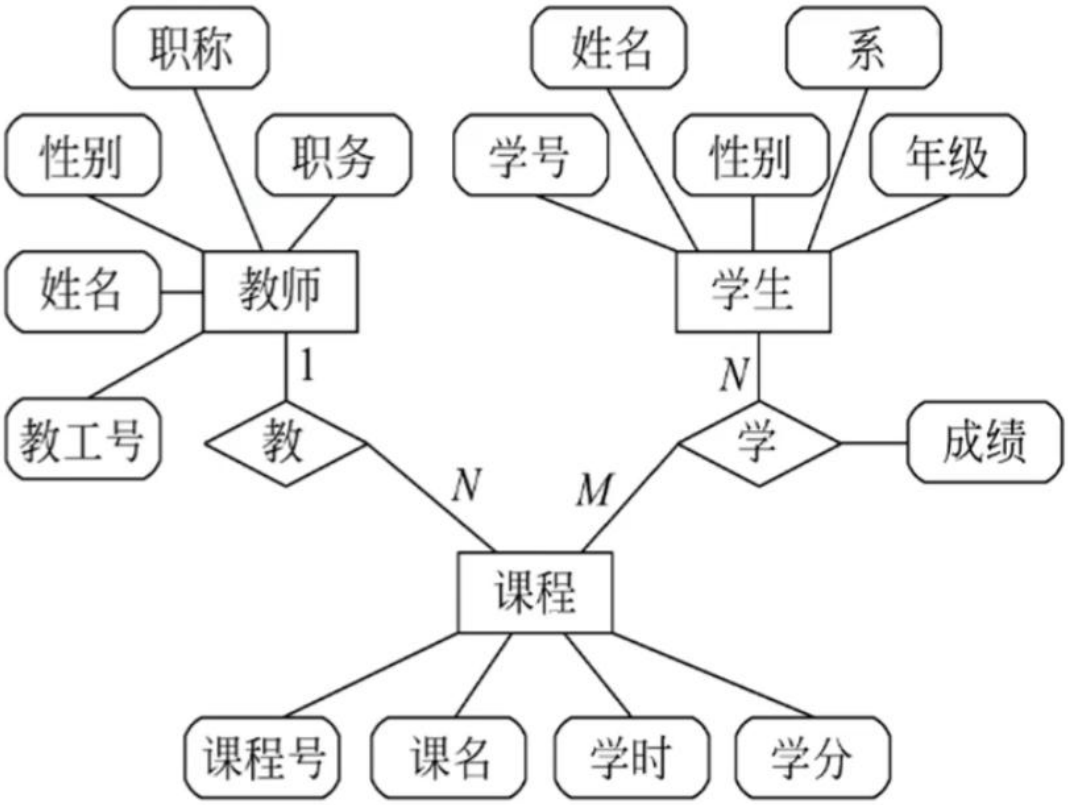
（3）E-R 模型的优点

E-R 模型比较接近人的思维习惯方式

E-R 模型使用简单的图形符号表达，便于用户理解

（4）示例

一个学生可选修多门课，一门课有若干学生选修;一个教师可讲授多门课，一门课只有一个教师讲授;学生选修一门课，产生成绩;学生的属性有学号、姓名等;教师的属性有教师编号，教师姓名等;课程的属性有课程号、课程名等。请画出该系统 E-R 图



第三节：状态转换图

一：定义（了解）

状态转换图(状态图) ：通过描绘系统的状态及引起系统状态转换的事件来表示系统的行为。状态图还提供了 行为建模机制，指明了作为特定事件的结果系统将做哪些动作

状态 ：状态是任何可以被观察到的系统行为模式，一个状态代表系统的一种行为模式。状态规定了系统对事件的响应方式。系统对事件的响应，既可以是做一个(或一系列)动作，也可以是仅仅改变系统本身的状态，还可以是既改变状态，又做动作

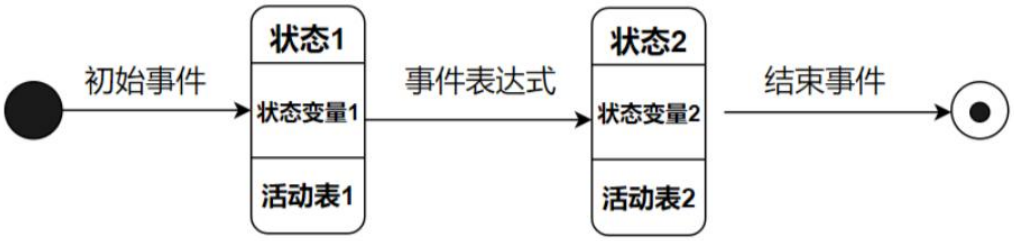
状态有初态、终态和中间状态

一张状态图只能有一个初态，而终态可以没有也可以有多个

事件：事件是在某个特定时刻发生的事情，它是对引起系统做动作或(和)从一个状态转换到另一个状态的外界事件的抽象。简而言之，事件就是引起系统做动作或(和)转换状态的控制信息

二：符号表示

- 初态：实心圆 ●
- 终态：同心圆，内为实心 ◉
- 状态：圆角矩形



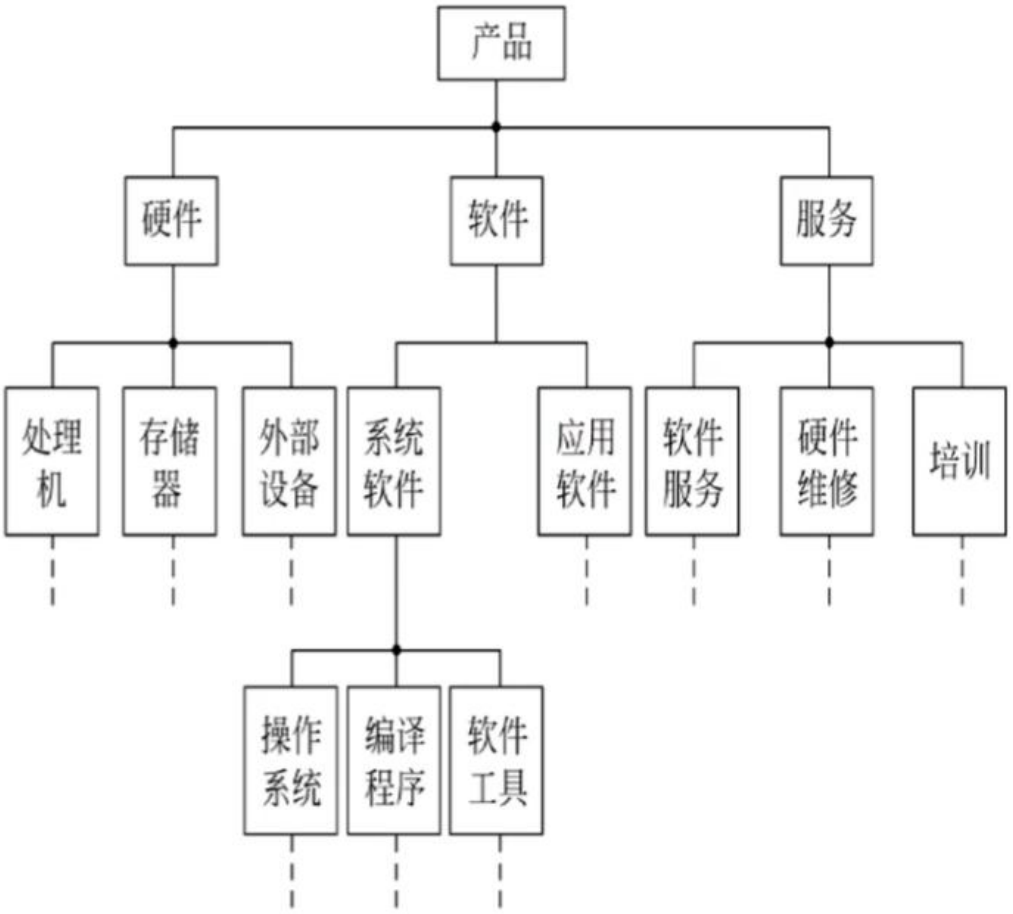
对于圆角矩形，可以将其分为上、中、下三部分：上部分是状态的名称；中部分是状态变量的名字和值；下部分是活动表

第四节：其他图形工具

一：层次方框图

用树形结构的一系列矩形框描绘数据的层次结构

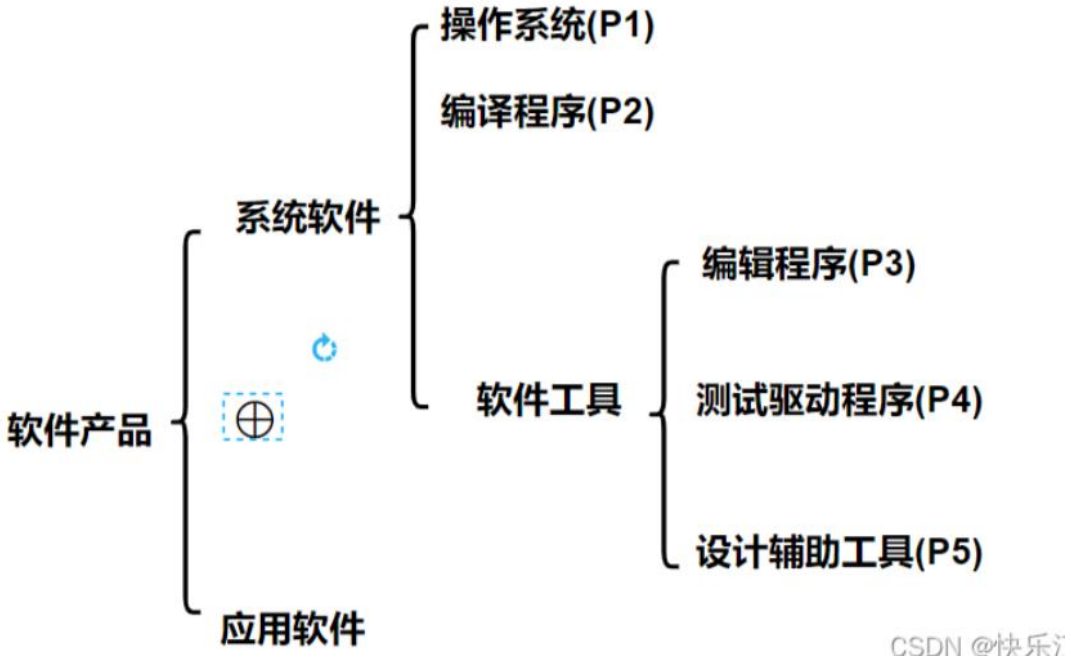
优点：随着结构的逐步精细，对数据结构的描绘也越来越详细



二：Warnier 图

用树形结构描绘信息的层次结构

优点：可以表明信息的逻辑组织；可以表明某类信息出现的条件或者是否重复出现



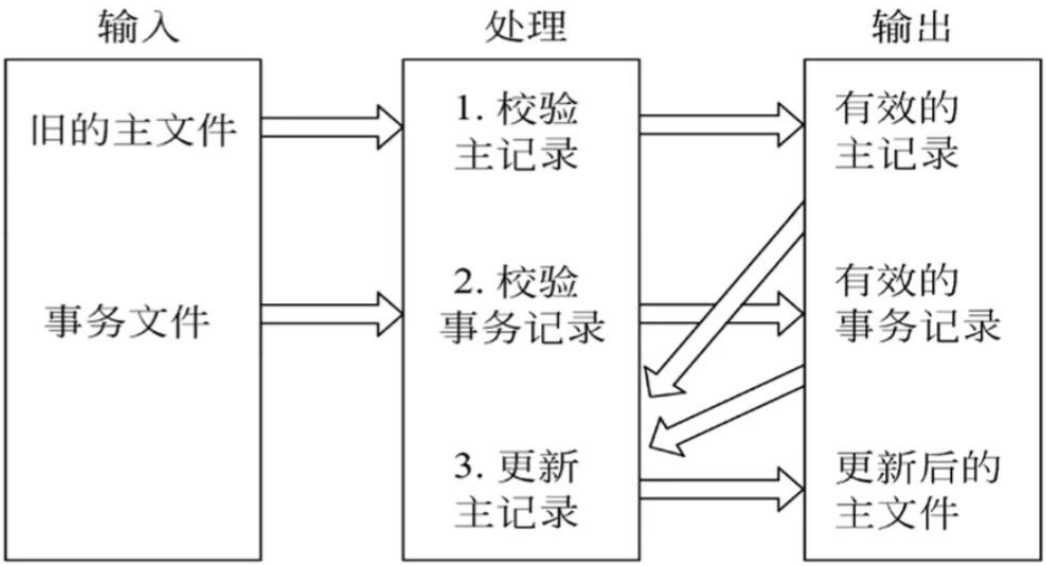
CSDN @快乐

⊕（异或）表明一类信息或一个数据元素在一定条件下才出现，而且这个符号上、下方的两个名字所代表的数据只能出现一次

圆括号中的数字指明了这个名字代表的信息类在这个数据结构中重复出现的次数

三：IPO 图

IPO 是输入、处理、输出图的简称，能够方便地描绘输入数据、对数据的处理和输出数据之间的关系



第五节：验证软件需求

一：验证需求正确性的四个方面

① 一致性：所有需求必须是一致的，任何一条需求不能和其他需求互相矛盾

② 完整性：需求必须是完整的，规格说明书应该包括用户需要的每一个功能或性能

③ 现实性：指定的需求应该能用现有的硬件和软件技术可以实现

④ 有效性：必须证明需求是正确有效的，确实能解决用户面对的问题

二：验证软件需求的方法

① 验证一致性：自然语言描述需求、形式化语言描述需求、使用软件工具验证

② 验证现实性：参照开发经验

③ 验证完整性和有效性：建立软件原型

三：用于需求分析的软件工具

（1）要求（了解）

① 必须有形式化的语法(或表)，因此可以用计算机自动处理使用这种语法说明的内容

② 使用这个软件工具能够导出详细的文档

③ 必须提供分析(测试)规格说明书的不一致性和冗余性的手段，并且应该能够产生一组报告指明对完整性分析的结果

④ 使用这个软件工具之后，应该能够改进通信状况

（2）PSL/PSA 系统

① PSL（问题陈述语言）：是用来描述系统的形式语言

② PSA（问题陈述分析程序）：是处理 PSL 描述的分析程序

③ 用 PSL 描述的系统属性放在一个数据库中。一旦建立起数据库之后即可增加信息、删除信息或修改信息，并且保持信息的一致性。PSA 对数据库进行处理以产生各种报告，测试不一致性或遗漏，并且生成文档资料。

第四章：总体设计

第一节：总体设计基本概念和设计过程

一：总体设计的概念

（1）定义

总体设计的基本目的就是回答“系统应该如何实现”这个问题，又称为概要设计或初步设计

（2）主要任务

划分出组成系统的物理元素程序、文件、数据库、人工过程和文档等，但是每个物理元素仍然处于黑盒子级，这些黑盒子里的具体内容将在以后仔细设计

设计软件的结构，也就是要确定系统中每个程序是由哪些模块组成的，以及这些模块相互间的关系

（3）步骤

寻找实现目标系统的各种不同的方案，需求分析阶段得到的数据流图是设想各种可能方案的基础

分析员从这些供选择的方案中选取若干个合理的方案，为每个合理的方案都准备一份系统流程图,列出组成系统的所有物理元素，进行成本/效益分析，并且制定实现这个方案的进度计划

进行必要的数据库设计，确定测试要求并且制定测试计划

（4）必要性（了解）

可以站在全局高度上，花较少成本，从较抽象的层次上分析对比多种可能的系统实现方案和软件结构，从中选出最佳方案和最合理的软件结构，从而用较低成本开发出较高质量的软件系统。

二：设计过程

设计过程包括系统设计阶段和结构设计阶段

（1）系统设计阶段

1：设想供选择的方案

在总体设计阶段应该考虑各种可能的实现方案，并且力求从中选出最佳方案。在总体设计阶段开始时只有系统的逻辑模型，分析员有充分的自由分析比较不同的物理实现方案，一旦选出了最佳的方案，将能大大提高系统的性能/价格比需求分析阶段得出的数据流图是总体设计的极好的出发点

常用的方法是：

① 设想把数据流图中处理分组的各种可能的方法

② 抛弃在技术上行不通的分组方法

③ 余下的分组方法代表可能的实现策略，并且可以启示供选择的物理系统

2：选取合理的方案

应该从前一步得到的一系列供选择的方案中选取若干个合理的方案,通常至少选取低成本、中等成本和高成本的 3 种方案

对于每个合理的方案，都应该准备下列 4 份材料

① 系统流程图

② 组成系统的物理元素清单

③ 成本/效益分析

④ 实现这个系统的进度计划

3：推荐最佳方案

应该综合分析对比各种合理方案的利弊，推荐一个最佳的方案，并且为推荐的方案制定详细的实现计划。用户和有关的技术专家应该认真审查分析员所推荐的最佳系统，如果该系统确实符合用户的需要,并且是在现有条件下完全能够实现的，则应该提请使用部门负责人进一步审批。在使用部门的负责人也接受了分析员所推荐的方案之后，将进入总体设计过程的下一个重要阶段一结构设计阶段

（2）结构设计阶段

4：功能分解

为确定软件结构，需要从实现角度把复杂的功能进一步分解。需要结合算法描述仔细分析数据流图中的每个处理，如果一个处理的功能过分复杂，必须把它的功能适当地分解成一系列比较简单的功能

5：设计软件结构

把模块组织成良好的层次系统,顶层模块调用它的下层模块以实现程序的完整功能，每个下层模块再调用下层的模块，从而完成程序的一个子功能，最下层的模块完成最具体的功能。软件结构，即由模块组成的层次统可以用层次图或结构图来描绘

6：设计数据库

对于需要使用数据库的那些应用系统，应该在需求分析阶段所确定的系统数据需求的基础上，进一步设计数据库

7：制定测试计划

在软件开发的早期阶段考虑测试问题，能促使软件设计人员在设计时注意提高软件的可测试性

8：书写文档

主要有以下几种：

① 系统说明

② 用户手册

③ 测试计划

④ 详细实现计划

⑤ 数据库设计结果

9：审查和复查

最后对总体设计结果进行严格的技术审查，在技术审查通过后再由客户从管理角度进行复审

第二节：设计原理

一：模块化

（1）模块

模块：模块是由边界元素限定的相邻程序元素的序列,而且有一个总体标识符代表它。模块是构成程序的基本构件。过程、函数、子程序和宏等，都可作为模块。面向对象方法学中的对象是模块，对象内的方法也是模块

（2）模块化

模块化：模块化就是把程序划分成独立命名且可独立访问的模块,每个模块完成一个子功能,把这些模块集成起来构成一个整体，可以完成指定的功能满足用户的需求。模块化是为了使一个复杂的大型程序能被人的智力所管理，是软件应该具备的唯一属性

（3）优点或作用

① 使软件结构清晰，不仅容易设计也容易阅读和理解
② 使软件容易测试和调试，有助于提高软件的可靠性
③ 提高软件的可修改性
④ 有助于软件开发工程的组织管理

二：抽象（了解）

抽象：抽出事物的本质特性而暂时不考虑它们的细节

三：逐步求精

（1）定义

逐步求精：逐步求精是软件工程技术的基础，为了能集中精力解决主要问题而尽量推迟对问题细节的考虑

（2）MIller 法则：注意力集中在(7 ± \pm± 2)

这是对人类能力的研究得到的结果：一个人能力的极限也只能把注意力集中在7±2 个信息块。这就像人弹跳的极限，人忍受肌饿的极限一样，是不可改变的客观因素。它很大程度上限制人的思维能力。这个法则被成为：Miller 法则我更喜欢这样来描述这个法则：一个人的注意力的极限是集中注意力在 7±2 个事物上。我们知道，相比安静的环境，在吵闹的环境中，我们比较难以集中注意力，所以我这里使用“事物”来代替“信息块”，因为我不想把吵闹声成为信息。另外还有一种定义是：一个人在任何时候只能把注意力集中在（7±2）个知识块上。这说明了一个普通人（特异人士除外）能同时学习的能力是有限的，所以这个法则对于我们如何分配工作和学习的精力，也提供了很好的指导和参考

四：信息隐藏和局部化（了解）

信息隐藏：指一个模块内包含的信息对于不需要这些信息的模块来说是不能访问的，主要是指模块的实现细节

局部化：指把一些关系密切的软件元素物理地放得彼此接近，有助于实现信息隐藏

在模块中使用局部数据元素是一个典型的例子

五：模块独立

（1）定义

模块独立：开发具有独立功能而且和其他模块之间没有过多的相互作用的模块，就可以做到模块独立。使得每个模块完成一个相对独立的特定子功能，并且和其他模块之间的关系很简单。模块独立的概念是模块化、抽象、信息隐藏和局部化概念的直接结果。其质量标准是耦合和内聚

（2）重要性

具有独立的模块的软件比较容易开发出来

独立的模块比较容易测试和维护

（3）模块耦合及其分类

A：定义

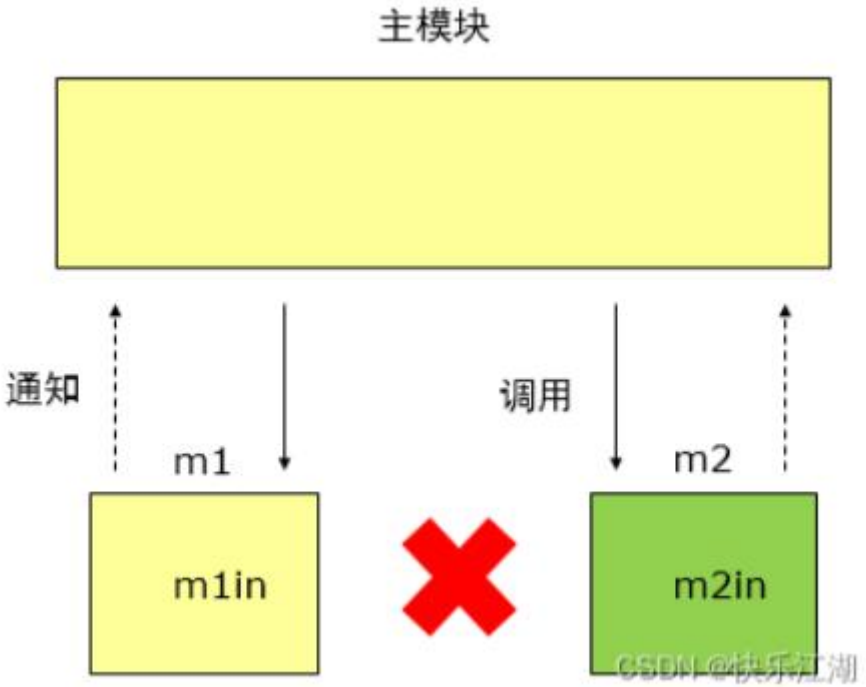
耦合：是对一个软件结构内不同模块间互连程序的度量。耦合强度取决于模块接口的复杂程度、通过接口的数据等。耦合度越高，模块独立性越弱

B：分类

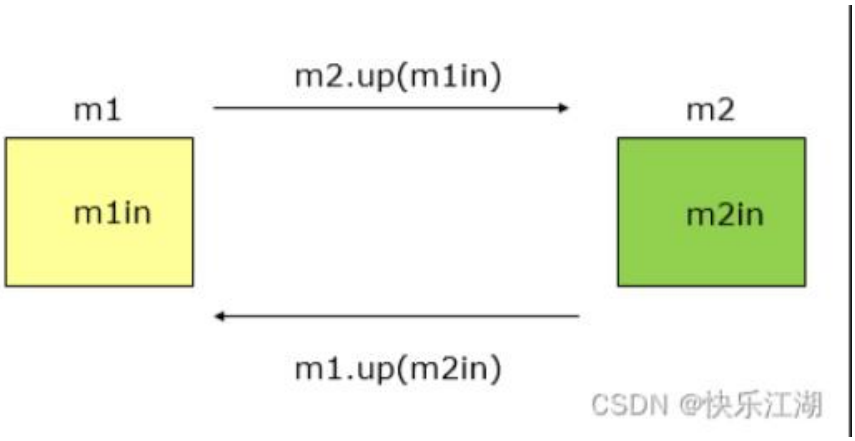
耦合度从低到高

① 完全独立
② 数据耦合
③ 特征耦合
④ 控制耦合
⑤ 外部耦合
⑥ 公共耦合
⑦ 内容耦合

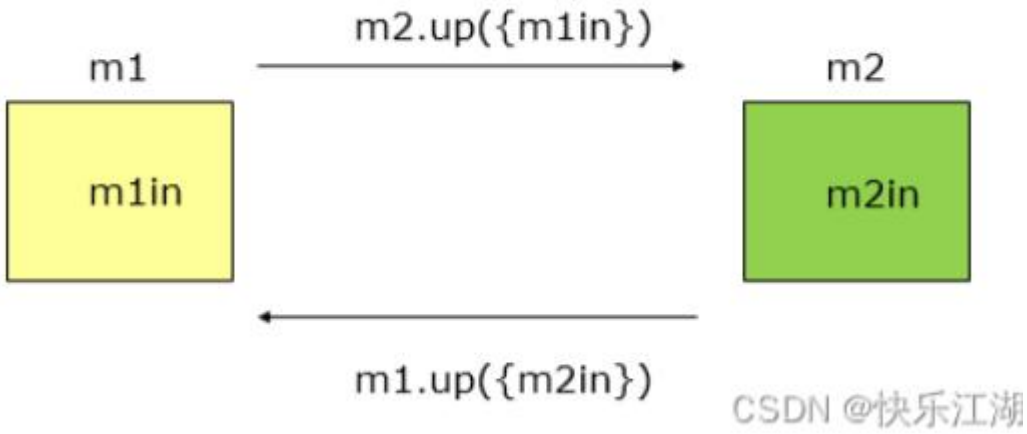
① 完全独立：如果两个模块中的每一个都能独立地工作而不需要另一个模块的存在，则称它们彼此完全独立，耦合程度最低。但是，在一个软件系统中不可能所有模块之间都没有任何连接



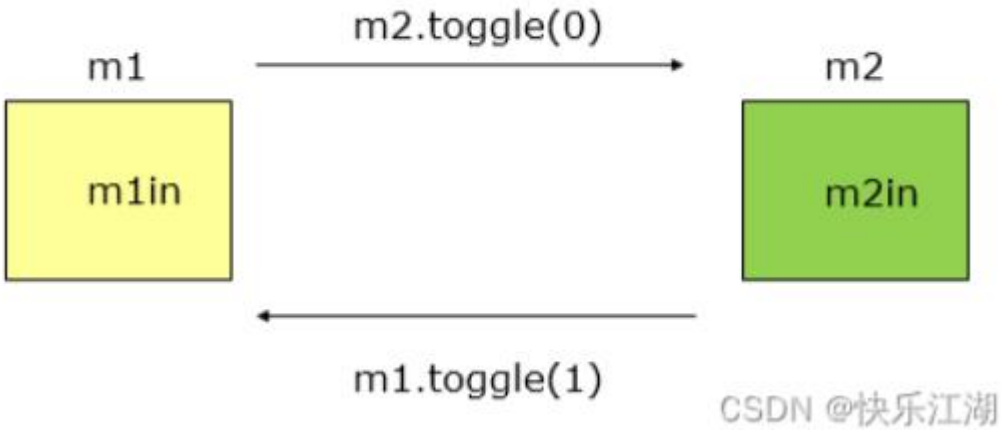
② 数据耦合： 如果两个模块彼此间通过参数交换信息，而且交换的信息仅仅是数据，则称它们是数据耦合。数据耦合是低耦合，系统中至少必须存在这种耦合



③ 特征耦合： 如果整个数据结构作为参数传递而被调用的模块只需要使用其中一部分数据元素,则称它们是特征耦合。在这种情况下，被调用的模块可以使用的数据多于它确实需要的数据，这将导致对数据的访问失去控制，从而给计算机犯罪提供了机会

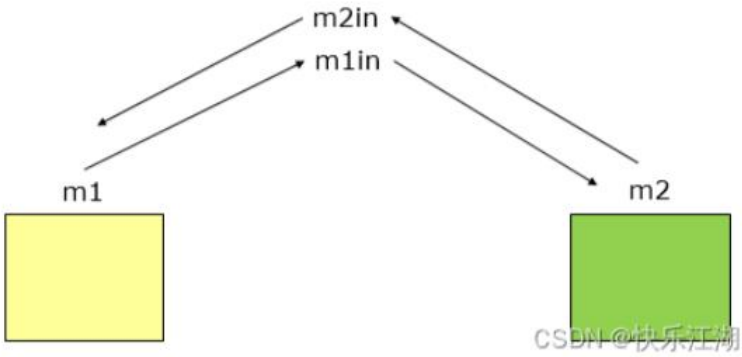


④ 控制耦合： 如果两个模块彼此间通过参数交换信息，并且传递的信息中包含控制信息(这种控制信息可以以数据的形式出现)，则称它们是控制耦合。控制耦合是中等程度的耦合，它增加了系统的复杂程度。控制耦合往往是多余的，可用数据耦合代替它

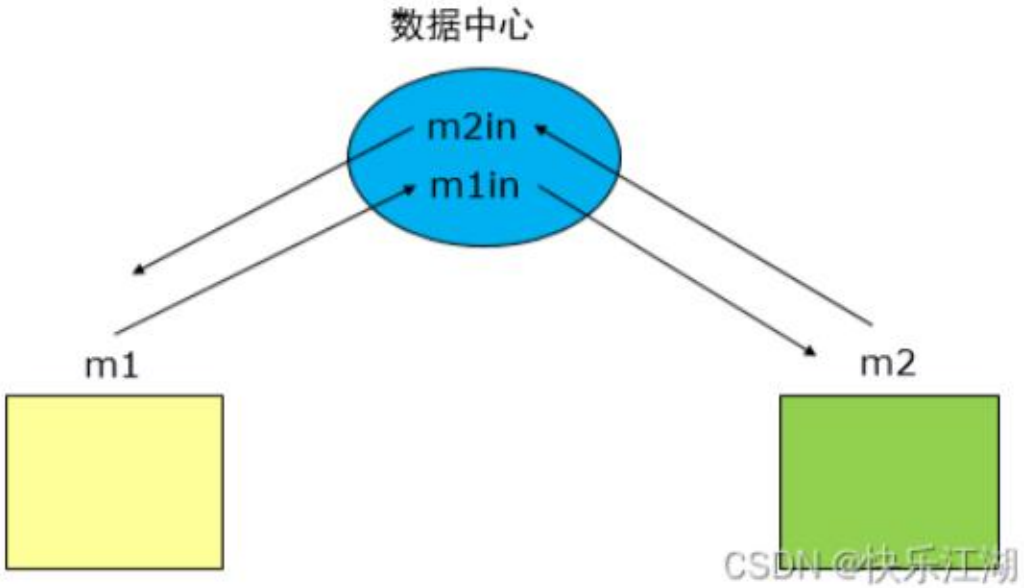


⑤ 外部耦合： 一组模块都访问同一全局简单变量，而且不通

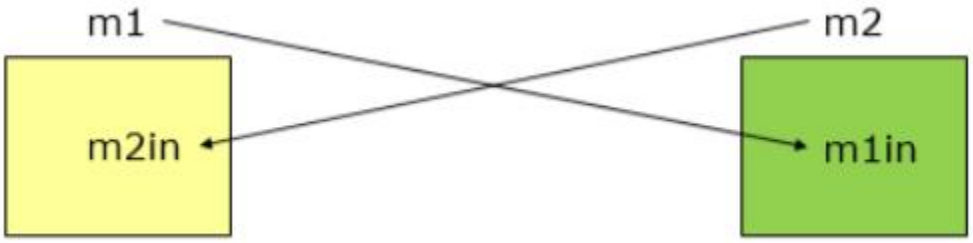
过参数表传递该全局变量的信息，则称之为外部耦合。外部耦合和公共耦合很像，区别就是一个是简单变量，一个是复杂数据结构



⑥ 公共耦合： 如果两个或多个模块通过一个公共数据环境相互作用,则称它们是公共环境耦合。公共环境耦合的复杂程度随耦合的模块个数增加而增加



⑦ 内容耦合： 内容耦合是最高程度的耦合，一个模块直接访问另一模块的内容，则称这两个模块为内容耦合。



C：设计原则

① 力求做到低耦合

② 尽量使用数据耦合

③ 少用控制耦合和特征耦合
④ 限制公共耦合的范围
⑤ 完全不用内容耦合

（4）模块内聚及其分类

A：定义

内聚：是用来度量一个模块内部各个元素彼此结合的紧密程度。内聚度越高，紧密程度越高

B：分类

内聚度从低到高。其中 1-3 属于低内聚；4-5 属于中内聚；6-7 属于高内聚

① 偶然内聚
② 逻辑内聚
③ 时间内聚
④ 过程内聚
⑤ 通信内聚
⑥ 顺序内聚
⑦ 功能内聚

① 偶然内聚：如果一个模块完成一组任务，这些任务彼此间即使有关系，关系也是很松散的，就叫做偶然内聚。有时在写完一个程序之后，发现一组语句在两处或多处出现，于是把这些语句作为一个模块以节省内存，这样就出现了偶然内聚的模块

② 逻辑内聚： 如果一个模块完成的任务在逻辑上属于相同或相似的一类， 则成为逻辑内聚

③ 时间内聚： 如果一个模块包含的任务必须在同一段时间内执行，就叫时间内聚。时间关系在一定程度上反映了程序的某些实质，所以时间内聚比逻辑内聚好一些

④ 过程内聚： 如果一个模块内的处理元素是相关的，而且必须以特定次序执行，则称为过程内聚。使用程序流程图作为工具设计软件时，往往得到的是过程内聚的模块。

⑤ 通信内聚： 如果模块中所有元素都使用同一个输入数据和(或)产生同一个输出数据，则称为通信内聚

⑥ 顺序内聚： 如果一个模块内的处理元素和同一个功能密切相关，而且这些处理必须顺序执行，则称为顺序内聚。根据数据流图划分模块时，通常得到顺序内聚的模块

⑦ 功能内聚： 如果模块内所有处理元素属于一个整体，完成一个单一的功能，则称为功能内聚。功能内聚是最高程度的内聚。

软件设计目标

高内聚、低耦合

第三节：启发规则

启发规则有：

① 改进软件结构提高模块独立性
② 模块规模应该适中

③ 深度、宽度、扇入和扇出应适当
④ 模块的作用域应该在控制域之内
⑤ 力争降低模块接口的复杂程度
⑥ 设计单入口单出口的模块
⑦ 模块功能应该可以预测但要防止过分局限

（1）改进软件结构提高模块独立性

设计出软件的初步结构后，应该审查分析这个结构，通过模块分解或合并，力求降低耦合提高内聚。

（2）模块规模应该适中

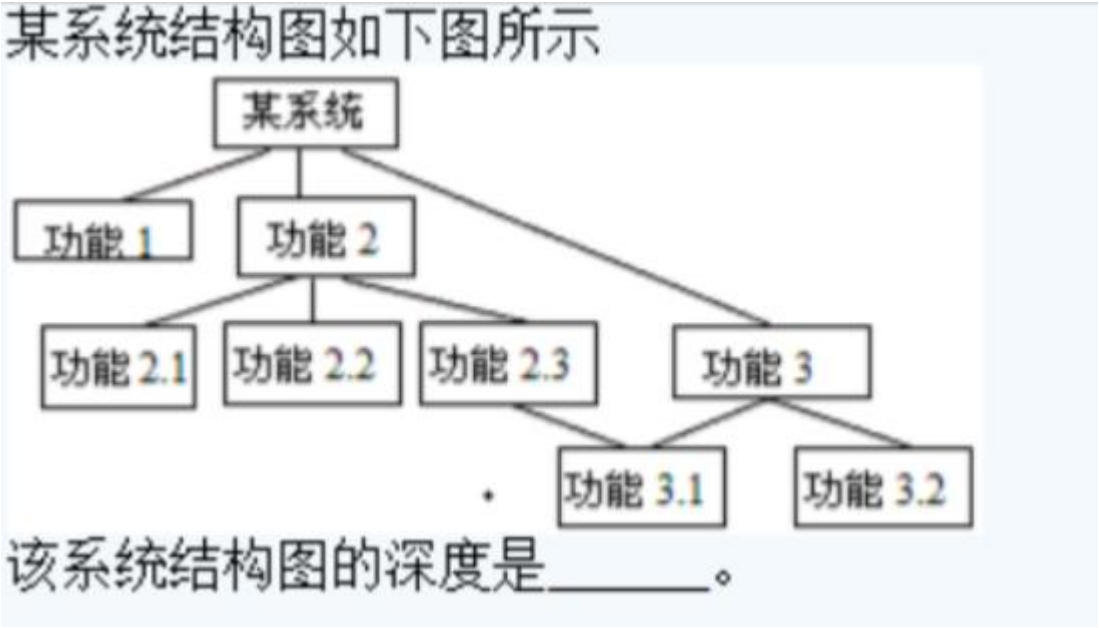
过大的模块往往是由于分解不充分，但是进一步分解必须符合问题结构，分解后不应该降低模块独立性

过小的模块开销大于有效操作，而且模块数目过多将使系统接口复杂。因此过小的模块有时不值得单独存在

（3）深度、宽度、扇入和扇出应适当

好的软件结构：顶层扇出较高，中层扇出较少，底层模块有高扇入

深度：表示软件结构中控制的层数，能粗略地标志一个系统的大小和复杂程度



深度为 3

可以按文件夹的的层级目录结构来对照理解，按照层级来看，

本道题中

第一层：某系统

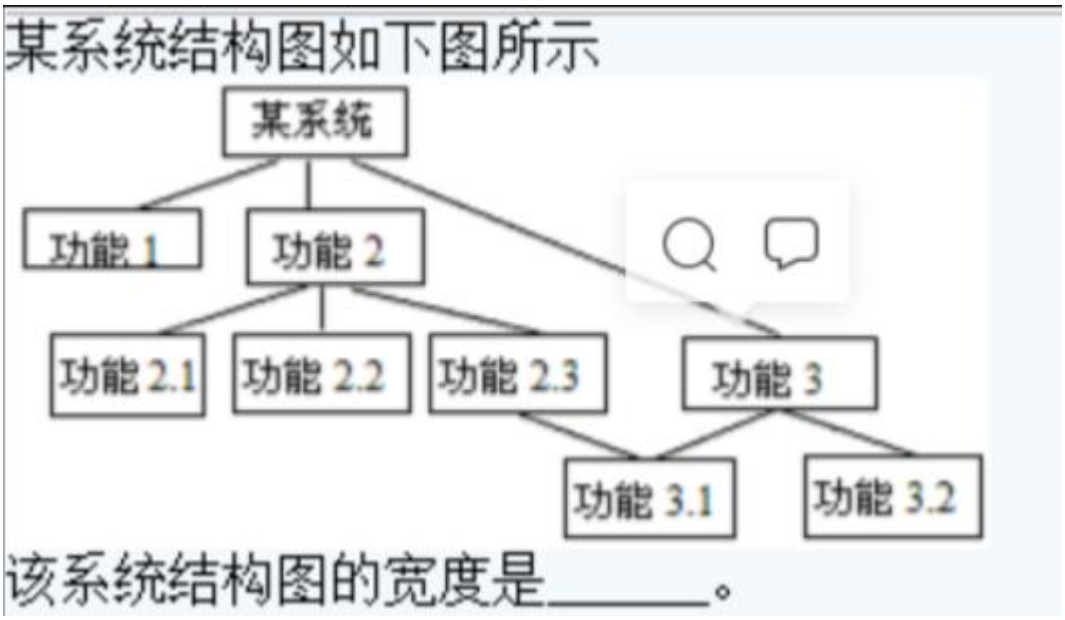
第二层：功能 1，功能 2，功能 3

第三层：功能 2.1，功能 2.2，功能 2.3，功能 3.1，功能 3.2

（注意：这里功能 3.1 和功能 3.2 其实是同一个层级的，类比文件目录结构不难看出），

深度即为这里分析的层数，所以这题的深度为 3。

宽度：宽度是软件结构内同一个层次上的模块总数的最大值。宽度越大系统越复杂。对宽度影响最大的因素是模块的扇出



宽度为 5

第三层：功能 2.1，功能 2.2，功能 2.3，功能 3.1，功能 3.2

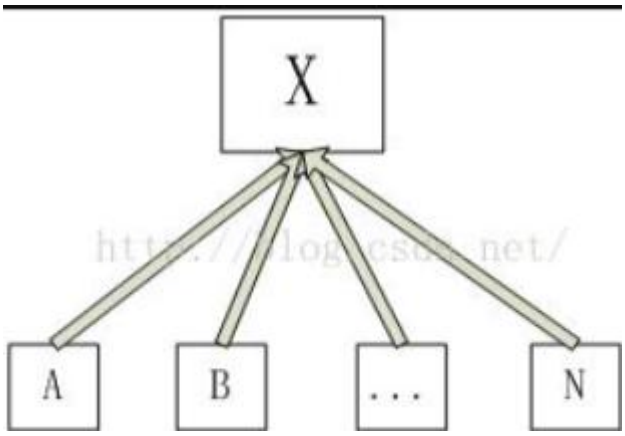
第三层的模块总数显然最大，为 5，

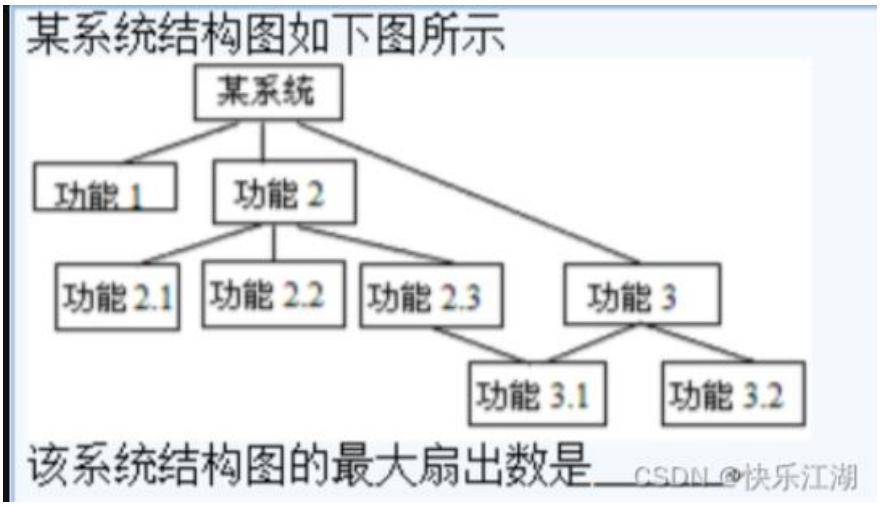
扇出：是一个模块直接控制的模块数目，设计扇出应注意

① 扇出过大意味着模块过分复杂，需要控制和协调过多的下级模块，扇出太大一般是因为缺乏中间层次,应适当增加中间层次的控制模块

② 扇出过小可把下级模块分解成若干个子功能模块，或合并到它的上级模块中去。分解模块或合并模块必须符合问题结构，不能违背模块独立原理

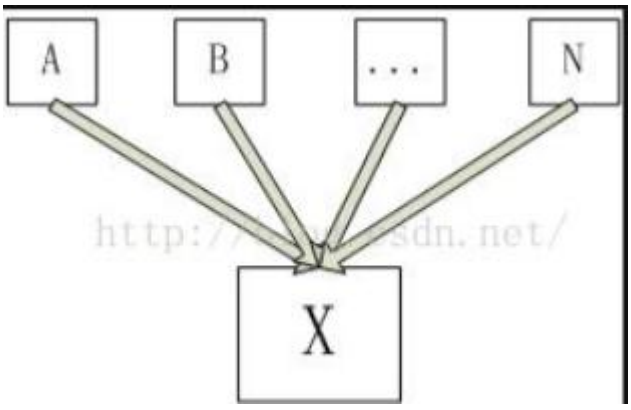
一个设计得好的典型系统的平均扇出通常是 3 或 4





最大扇出数是 3

扇入：表明有多少个上级模块直接调用它。扇入越大则共享该模块的上级模块数目越多。但是，不能违背模块独立原理单纯追求高扇入



（4）模块的作用域应该在控制域之内

A：定义

作用域：指受该模块内一个判定影响的所有模块的集合

控制域：是这个模块本身以及所有直接或者间接从属于它的模块的集合

B：规则

在一个设计得很好的系统中，所有受判定影响的模块应该都从属于做出判定的那个模块，最好局限于做出判定的那个模块本身及它的直属下级模块

C：修改方法（了解）

把做判定的点往上移

把在作用域内但不在控制域内的模块移到控制域内

（5）力争降低模块接口的复杂程度

模块接口复杂是软件发生错误的一个主要原因。应该仔细设计模块接口，使得信息传递简单并且和模块的功能一致。接口复杂或不一致(即看起来传递的数据之间没有联系)是紧耦合或低内聚的征兆，应该重新分析这个模块的独立性

（6）设计单入口单出口的模块

这条规则警告软件工程师不要使模块间出现内容耦合。当从顶部进入模块

并且从底部退出来时，软件是比较容易理解的，因此也是比较容易维护的

（7）模块功能应该可以预测但要防止过分局限

模块的功能应该能够预测，但也要防止模块功能过分局限。

可预测：如果一个模块可以当做一个黑盒子，即只要输入的数据相同就产生同样的输出,这个模块的功能就是可以预测的。由于内部存储器对于上级模块而言是不可见的，所以这样的模块既不易理解又难于测试和维护

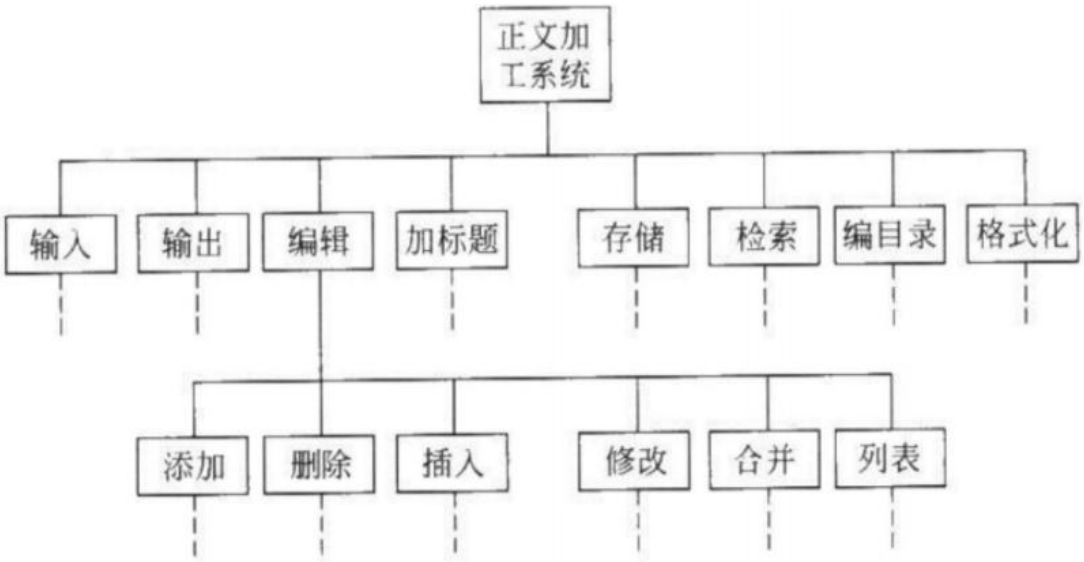
过分局限：如果一个模块任意限制局部数据结构的大小，过分限制在控制流中可以做出的选择或者外部接口的模式，那么这种模块的功能就过分局限，使用范围也就过分狭窄了。

第四节：描绘软件结构的图形工具

一：层次图和 HIPO 图

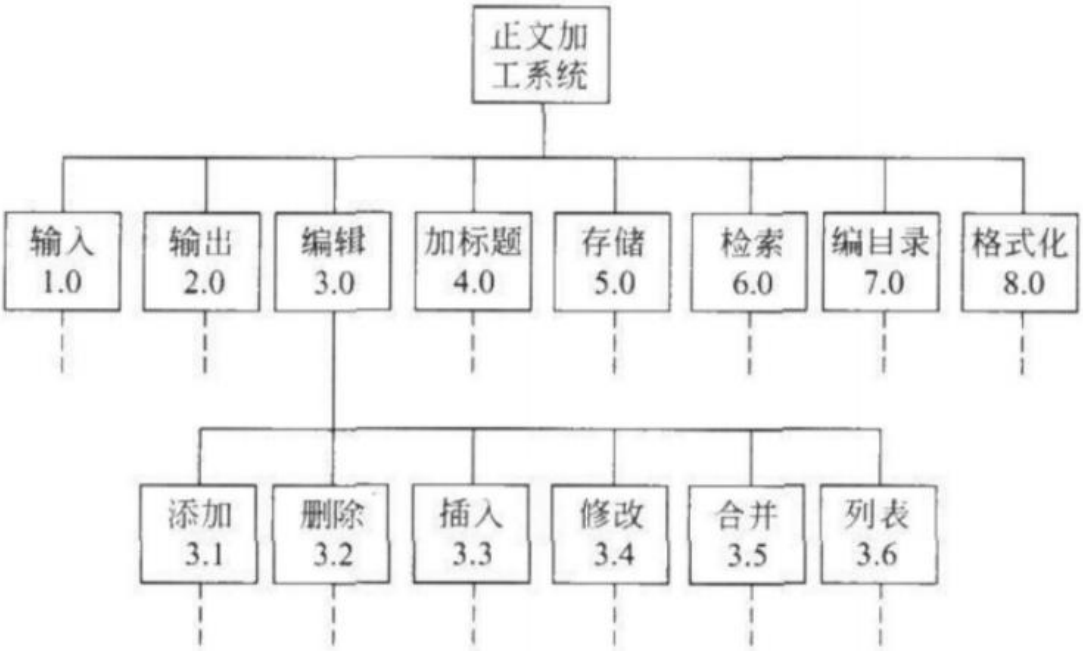
层次图用于描绘软件的层次结构，其中一个矩形框代表一个模块，方框间的连线表示调用关系而不像层次方框图那样表示组成关系

- 层次图很适于在自顶向下设计软件的过程中使用。通常用层次图作为描绘软件结构的文档



- 最顶层的方框代表正文加工系统的主控模块，它调用下层模块完成正文加工的全部功能;第二层的每个模块控制完成正文加工的一个主要功能，第二层的模块又可以调用下一层的模块完成具体的工作

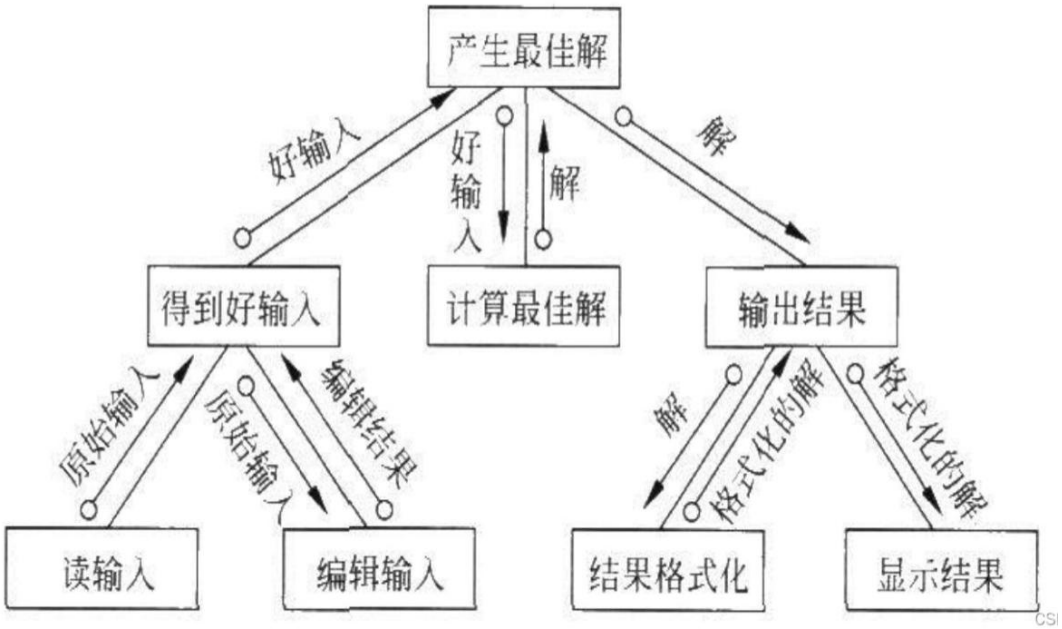
HIPO 图本质就是层次图加编号



二：结构图

（1）定义

结构图不仅描述调用关系，还描述传递的信息和调用方式



（2）符号

A：基本符号

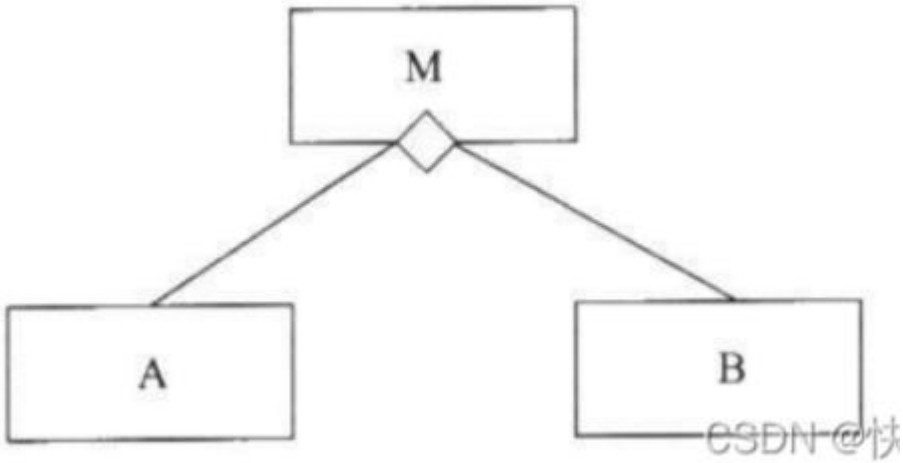
方框代表模块、框内注明模块的名字或主要功能

箭头或直线表示调用关系

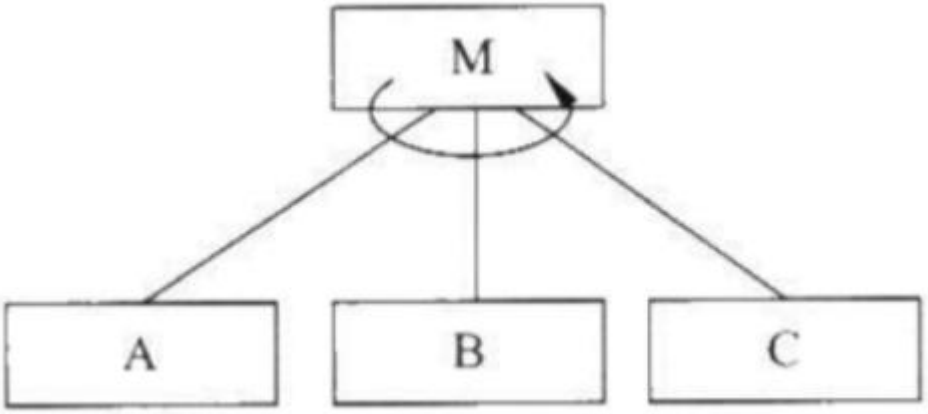
尾部是空心圆表示传递的是数据；若是实心圆则表示传递的是控制信息

B：特殊符号

表示当模块 M 中某个判定为真时调用模块 A，为假时调用模块 B



表示模块 M 循环调用模块 A、B 和 C



第五章：详细设计

第三节：过程设计工具

描述程序处理过程的工具称之为过程设计工具，可以分为图形、表格和语言三类，具体有

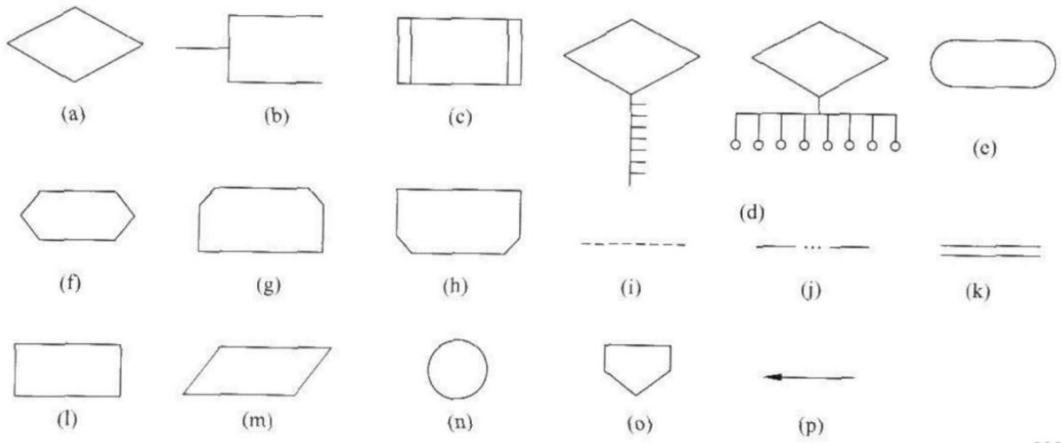
程序流程图

① 盒图（N-S 图）
② PAD 图
③ 判定表
④ 判定树
⑤ 过程设计语言（PDL）

（1）程序流程图

A：符号

程序流程图又称为程序框图，是历史最悠久，使用最广泛的描述过程设计的方法，然而它也是用得最混乱的一种方法，涉及符号如下



a：选择 b：注释 c：预先定义的处理 d：多分支 e：开始或停止 f：准备 g：循环上界限 h：循环下界限 i：虚线 j：省略符 k：并行方式 l：处理 m：输入输出 n：连接 o：换页连接 p：控制流

B：优缺点

优点：

对控制流程的描绘很直观，便于初学者掌握

缺点：

① 程序流程图本质上不是逐步求精的好工具，它诱使程序员过早地考虑程序的控制流程，而不去考虑程序的全局结构

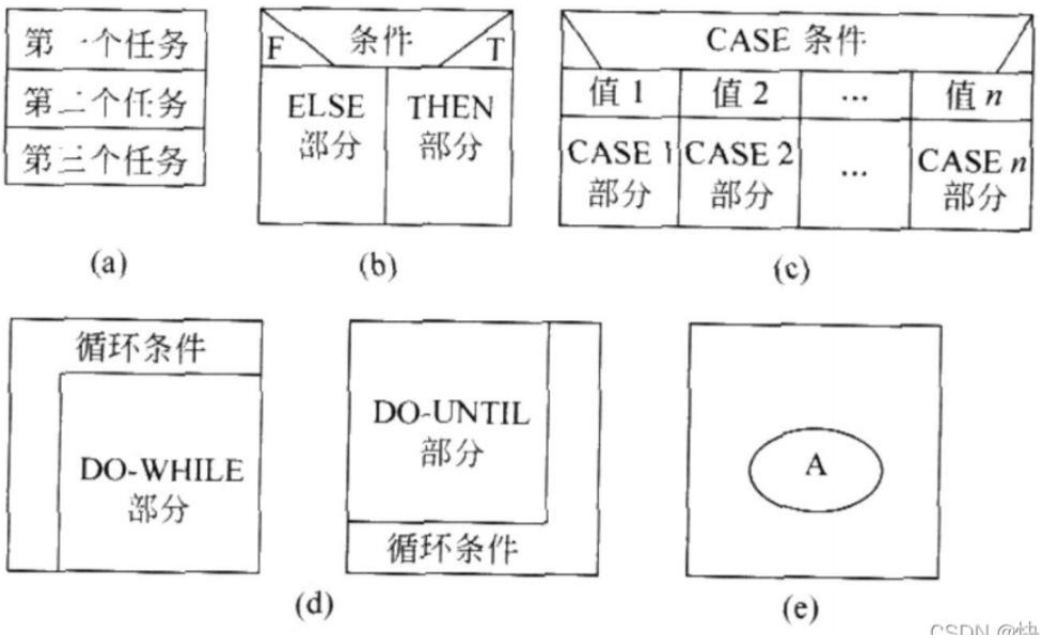
② 程序流程图中用箭头代表控制流，因此程序员不受任何约束，可以完全不顾结构程序设计的精神，随意转移控制

③ 程序流程图不易表示数据结构

（2）盒图（N-S）

A：符号

出于要有一种不允许违背结构程序设计精神的图形工具的考虑，提出了盒图，又称为 N-S 图。其基本符号如下



a：顺序结构 b：IF_TEEN_ELSE 型分支 c：CASE 型多分支
d：循环结构 e：调用子程序 A

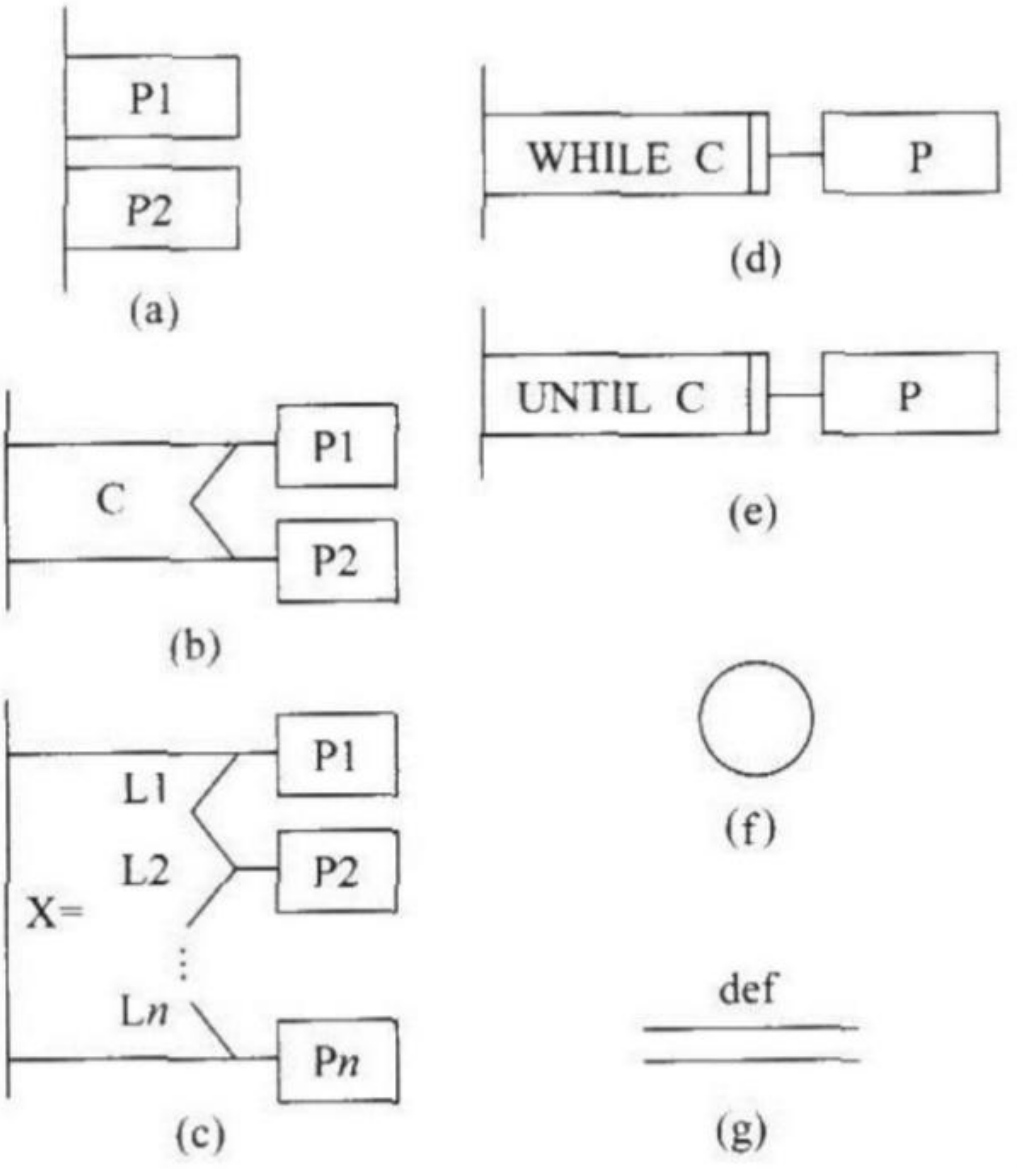
B：优点（了解）

① 功能域明确，可以从盒图上一眼就看出来
② 不可能任意转移控制
③ 很容易确定局部和全程数据的作用域
④ 很容易表现嵌套关系，也可以表示模块的层次结构

（3）PAD 图

A：符号

PAD 是问题分析图（problem analysis is diagram）的英文缩写，是使用二维树形结构的图来表示程序的控制流，这种图翻译为程序代码比较容易。其基本符号如下

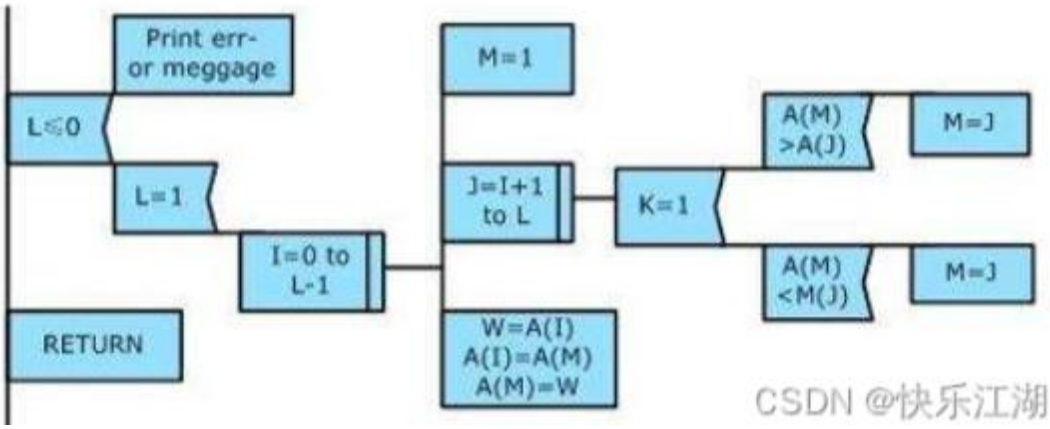


a：顺序 b：选择（IF C THEN P1 ELSE P2） c：CASE 型多分支
d：WHILE 型循环（WHILE C DO P） e：UNTIL 型循环（REPEAT P UNTIL C）
f：语句符号 g：定义

B：优点

① 使用表示结构化控制结构的 PAD 符号所设计出来的程序必然是结构化程序

② PAD 图所描绘的程序结构十分清晰。图中最左面的竖线是程序的主线，即第一层结构。随着程序层次的增加，PAD 图逐渐向右延伸，每增加一个层次，图形向右扩展一条竖线。PAD 图中竖线的总条数就是程序的层次数



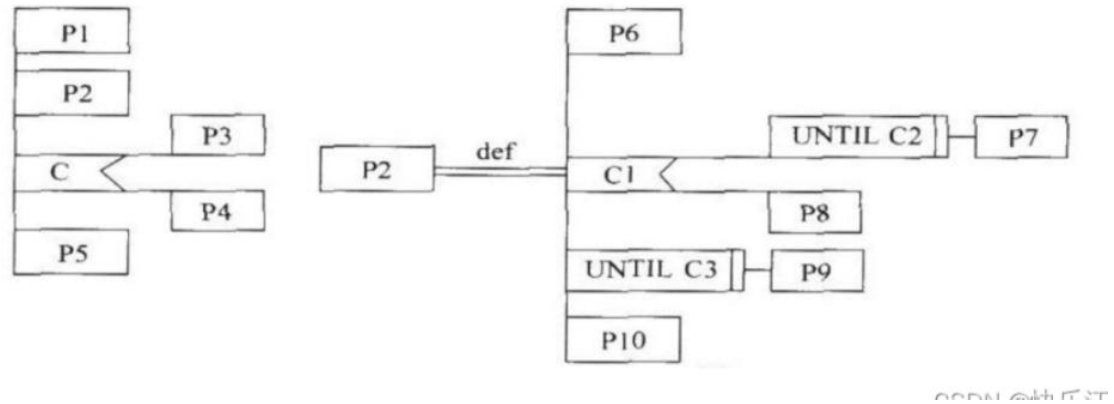
CSDN @快乐江湖

③ 用 PAD 图表现程序逻辑，易读、易懂、易记。PAD 图是二维树形结构的图形，程序从图中最左竖线上端的结点开始执行 ，自上而下，从左向右顺序执行，遍历所有结点

④ 容易将 PAD 图转换成高级语言源程序，这种转换可用软件工具自动完成，从而可省去人工编码的工作，有利于提高软件可靠性和软件生产率

⑤ 即可用于表示程序逻辑，也可用于描绘数据结构

⑥ PAD 图的符号支持自顶向下、逐步求精方法的使用。开始时设计者可以定义一个抽象的程序，随着设计工作的深入而使用 def 符号逐步增加细节，直至完成详细设计(如下图所示，左图表示初始的 PAD 图，右图表示使用 def 符号细化处理框 P2)



（4）判定表

A：组成

当算法中包含多重嵌套的条件选择时，判定表能够清晰地表示复杂的条件组合与应做的动作之间的对应关系。由以下部分组成

① 左上部列出所有条件

② 左下部是所有可能做的动作

③ 右上部是表示各种条件组合的一个矩阵

④ 右下部是和每种条件组合相对应的动作

例如：

假设某航空公司规定,乘客可以免费托运重量不超过 30kg（条件 1）的行李。当行李重量超过 30kg 时，对头等舱（条件 2）的国内乘客（条件 3） 超重部分每公斤收费 4 元，对其他舱（条件 2） 的国内乘客超重部分每公斤收费 6 元，对外国乘客（条件 4）超重部分每公斤收费比国内乘客多–倍，对残疾乘客超重部分每公斤收费比正常乘客少一半

在表的右上部分中 T 表示它左边那个条件成立，F 表示条件不成立，空白表示这个条件成立与否并不影响对动作的选择。判定表右下部分中画 X 表示做它左边的那项动作，空白表示不做这项动作

	规则								
	1	2	3	4	5	6	7	8	9
国内乘客		T	T	T	T	F	F	F	F
头等舱		T	F	T	F	T	F	T	F
残疾乘客		F	F	T	T	F	F	T	T
行李重量 $W \leqslant 30$kg	T	F	F	F	F	F	F	F	F
免费	×								
$(W-30)\times 2$				×					
$(W-30)\times 3$					×				
$(W-30)\times 4$		×						×	
$(W-30)\times 6$			×						×
$(W-30)\times 8$						×			
$(W-30)\times 12$							×		

CSDN @快!

[解析]只要行李重量不超过 30kg， 不论这位乘客持有何种机票，是中国人还是外国人，是残疾人还是正常人，一律免收行李费，这就是表右部第一列(规则 1)表示的内容。当行李重量超过 30kg 时，根据乘客机票的等级、乘客国籍及是否残疾人而使用不同算法计算行李费，这就是从规则 2 到规则 9 所表示的内容

B：优缺点

优点：

① 判定表能够简洁而又无歧义地描述处理规则

② 判定表和布尔代数或卡诺图结合起来使用，可以更加直观、简洁、清晰的描述规则

缺点：

① 不能同时清晰地表示出问题的顺序性和重复性

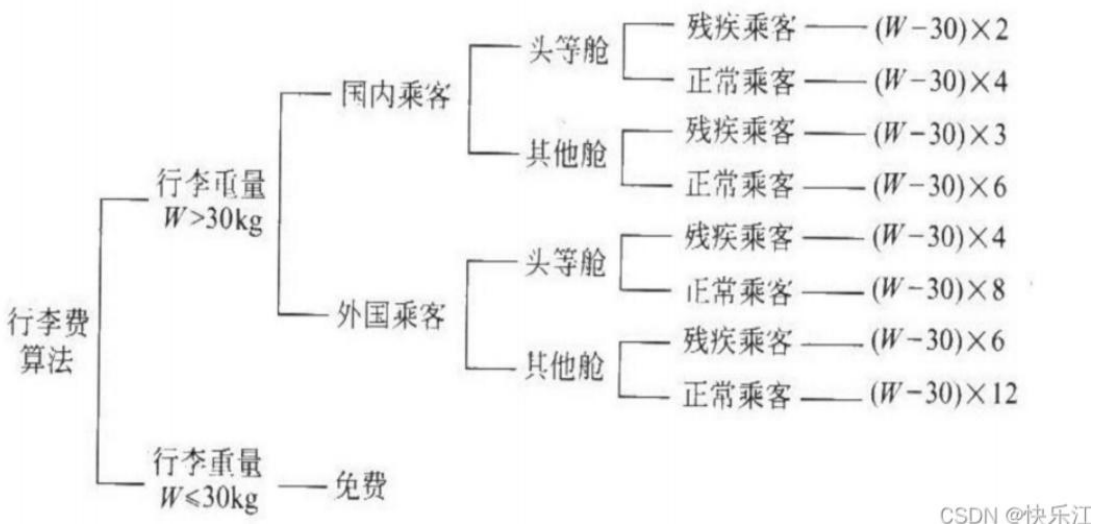
② 初次接触这种工具的人理解它需要有一个学习过程

③ 数据元素增多时，判定表的简洁程度大幅下降

（5）判定树

是判定表的变种，也能清晰地表示复杂的条件组合与应做的动作之间的对应关系

例如：



CSDN @快乐江

（6）过程设计语言（PDL）（了解）

PDL 也即伪代码，是用正文形式表示数据和处理过程的设计工具。PDL 具有严格的关键字外部语言，用于定义控制结构和数据结构。PDL 表示实际操作和条件的内部语言通常又是灵活自由的，可以适用各种工程项目的需要

第五节：程序复杂度的定量度量

一：价值（了解）

把程序的复杂程度乘以适当数可估算出软件中错误的数量以及开发需要的工作量

定量度量的结果可以用来比较两个不同的设计或两个不同算法的优劣

程序的定量的复杂程度可以作为模块规模的精确限度

二：McCabe 方法

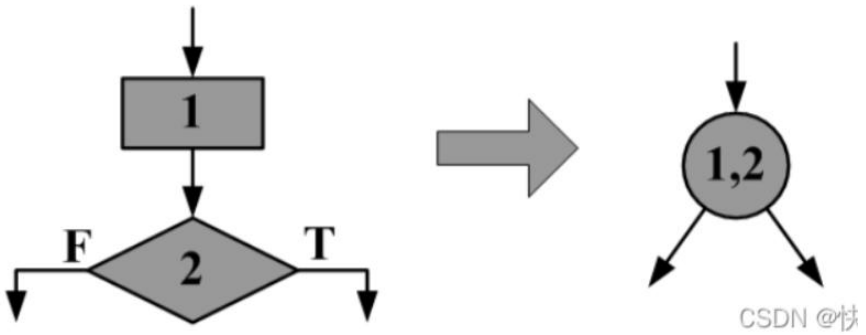
（1）流图

A：定义

流图实质上是“退化了的”程序流程图，它仅仅描绘程序的控制流程，完全不表现对数据具体操作以及分支或循环的具体条件，流图通常被用来突出表示程序的控制流

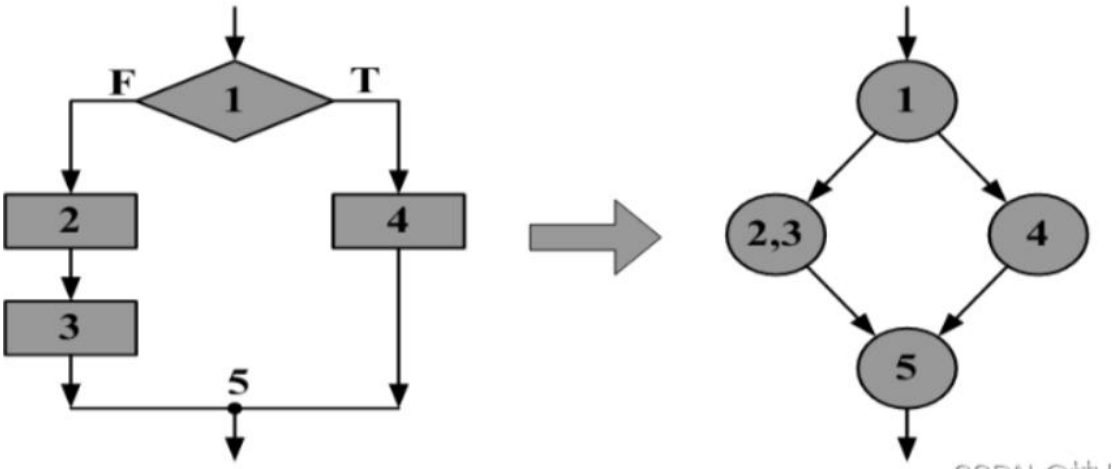
B：把程序流程图映射为流图

①：对于顺序结构，一个顺序处理和下一个选择可以映射为一个结点



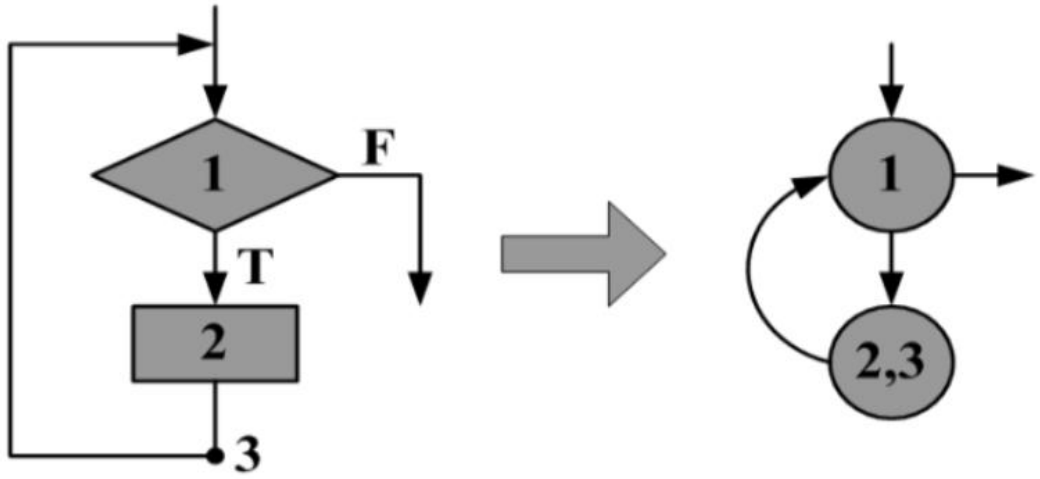
CSDN @快

②：对于选择语句，开始/结束语句映射为一个结点，两条分支至少各映射成一个结点

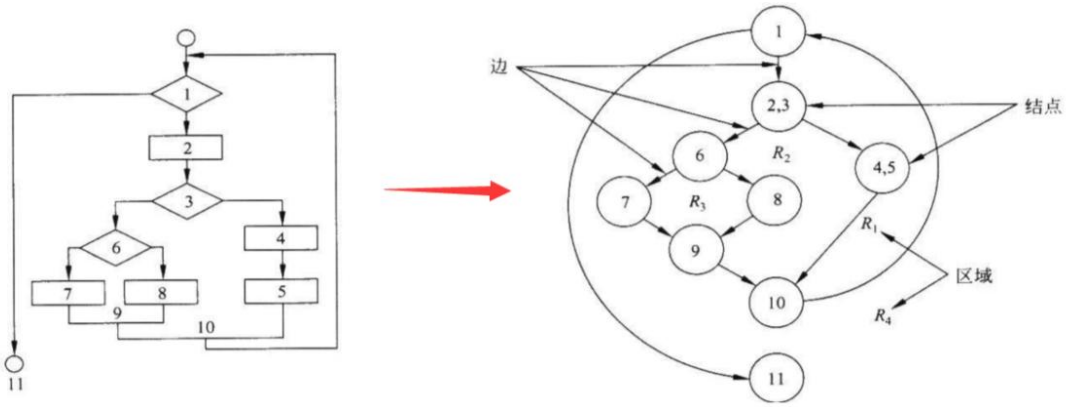


CSDN @快

③：开始语句和结束语句各映射成一个结点



CSDN @快乐

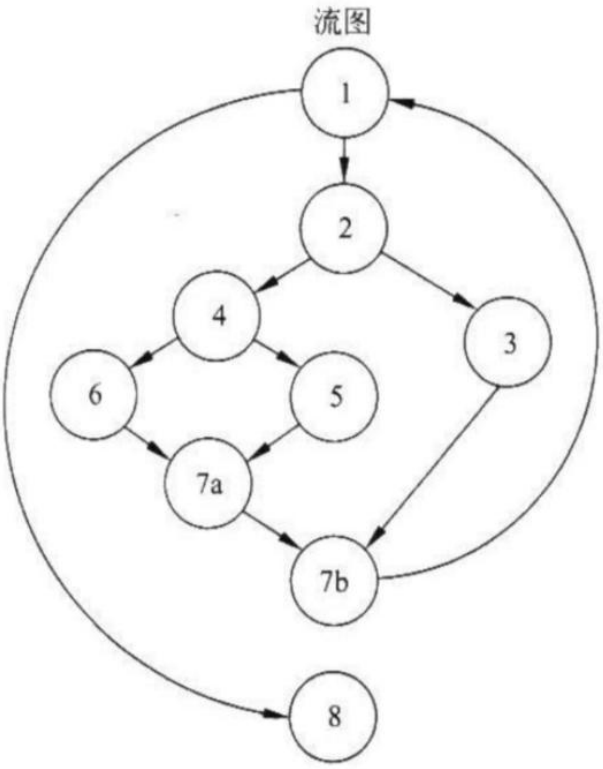


- 流图中用圆表示结点，一个圆代表一条或多条语句（比如 4、5）。程序流程图中的一个顺序的处理框序列和一个菱形判定框，可以映射成流图中的一个结点（比如 2,3）

- 流图中的箭头线称为边，代表控制流。流图中一条边必须终止于一个结点（比如 9 必须为一个结点），即使这个结点并不代表任何语句
- 由边和结点围成的面积称为区域，计算区域数时应包括图外部未被围起来的区域

C：PDL 翻译为流图

```
PDL
procedure:sort
1:   do while records remain
2:      read record;
        if record field 1=0
3:        then process record;
            store in buffer;
            increment counter;
4:      elseif record field 2=0
5:        then reset counter;
6:        else process record;
            store in file;
7a:     endif
        endif
7b: enddo
8:   end
```



（2）环形复杂度

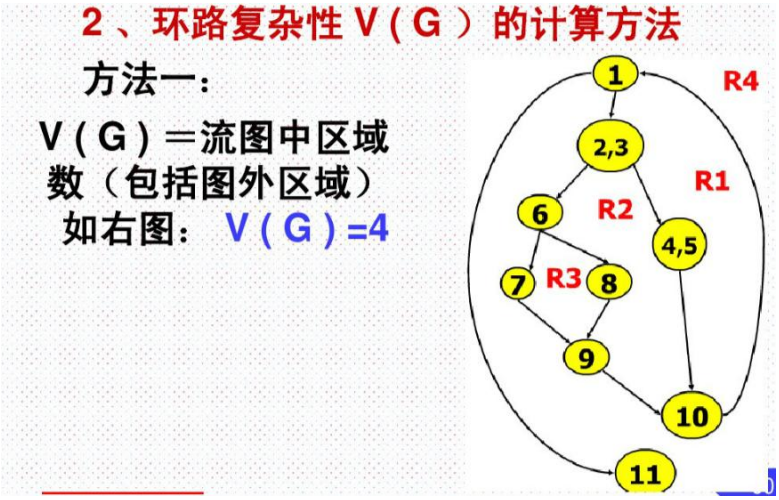
A：定义

McCabe 方法根据程序控制流的复杂程度定量度量程序的复杂程度，这样度量出的结果称为程序的环形复杂度

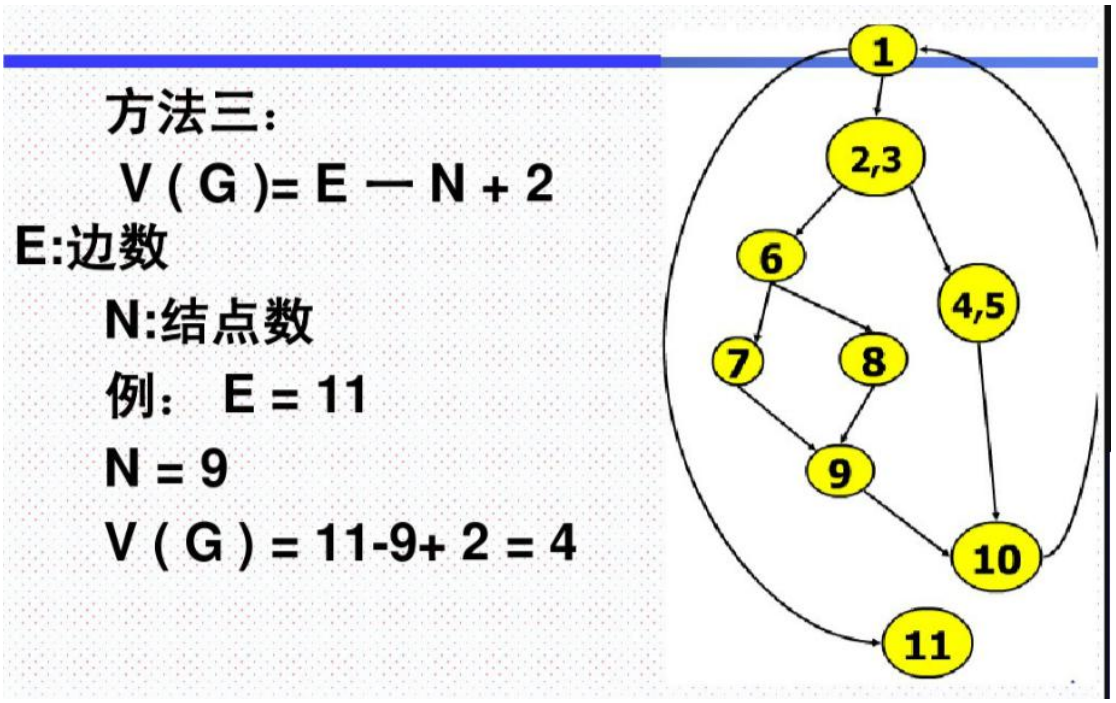
B：计算方法

环形复杂度定量度量程序的逻辑复杂度，可以用下述 3 种方法中的任何一种来计算环形复杂度:

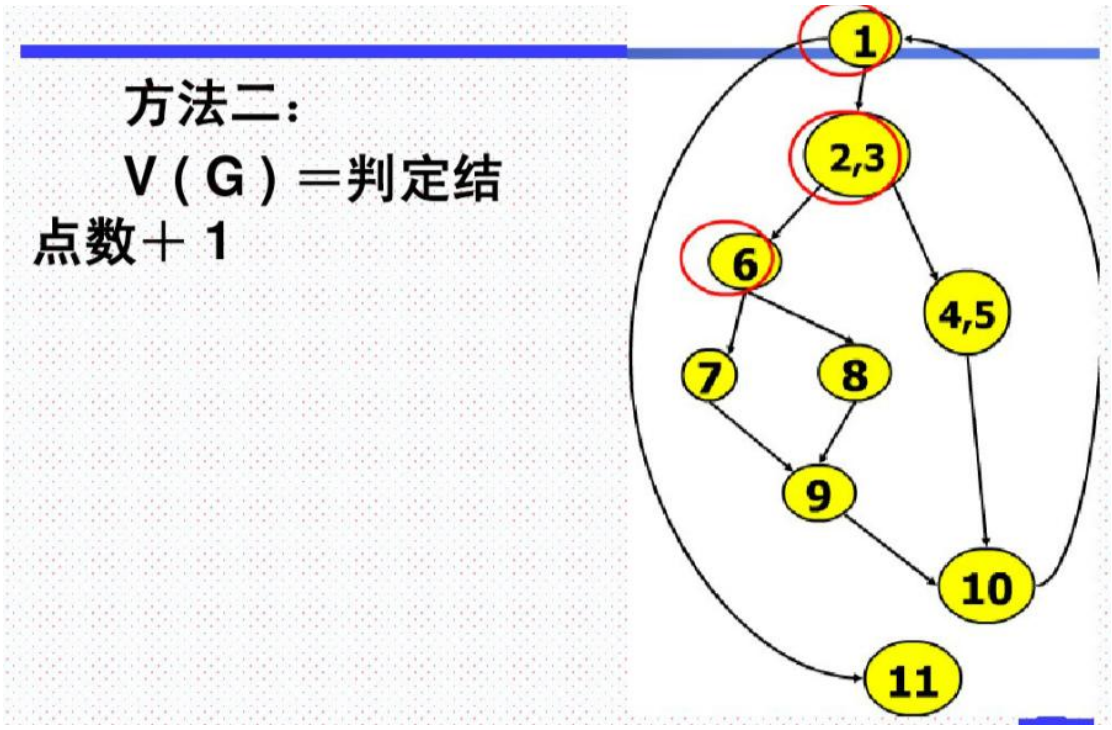
- 流图中的区域数等于环形复杂度



- 流图 G 的环形复杂度 V(G)=E-N+2,E 是流图中边的条数，N 是结点数



- 流图 G 的环形复杂度 V(G)=P+1，其中，P 是流图中判定结点的数目。V(G)小于等于 10 比较科学



三：Halstead 方法（了解）

（1）定义

Halstead 方法是根据程序中运算符和操作数的总数来度量程序的复杂程度

（2）方法

1.程序长度N定义，其中N_1是程序中运算符出现总次数，N_2是程序中操作数出现总次数

$$N = N_1 + N_2$$

2.预测程序长度的公式，使用的不同运算符的个数为n_1，不同操作数的个数为n_2

$$H = n_1 log_2 n_1 + n_2 log_2 n_2$$

3.预测程序中包含错误的个数的公式

$$E = N * log_2 (n_1 + n_2)/3000$$

第六章：实现和测试

第一节：编码

一：定义

编码是把软件设计结果翻译成用程序设计语言书写的程序,是对设计的进一步具体化，因此程序的质量主要取决于软件设计的质量

二 ：程序设计语言的选择

（1）重要性（了解）

① 程序设计语言的特点必然会影响人的思维和解题方式

② 程序设计语言会影响人和计算机通信的方式和质量

③ 程序设计语言会影响其他人阅读和理解程序的难易程度

（2）适宜的程序语言的优点

① 能使根据设计去完成编码时困难最少

② 可以减少需要的程序测试量

③ 可以得出更容易阅读和更容易维护的程序

（3）程序设计语言的选择标准

A：理想标准

① 选用高级语言编写程序

② 选用的高级语言应该有理想的模块化机制，以及可读性好的控制结构和数据结构

③ 选用语言特点应该使编译程序能够尽可能多地发现程序中的错误

④ 选用的高级语言应该有良好的独立编译机制

B：实际标准

①：系统用户的要求：如果所开发的系统由用户负责维护，用户通常要求用他们熟系的语言编写

②：可以使用的编译程序：运行目标系统的环境中可以提供的编译程序往往限制了可以选用的语言的范围

③：可以得到的软件工具：如果某种语言有支持程序开发的软件工具可以利用，则目标系统的实现和验证都变得比较容易

④：工程规模：如果工程规模很庞大，现有的语言又不完全适用，那么设计并实现一种供这个工程项目专用的程序设计语言，可能是一个正确的选择

⑤：程序员的知识：虽然对于有经验的程序员来说，学习一种新语言并不困难，但是要完全掌握一种新语言却需要实践。如果和其他标准不矛盾，那么应该选择一种已经为程序员所熟悉的语言

⑥：软件可移植性要求：如果目标系统将在几台不同的计算机上运行，或者预期的使用寿命很长，那么选择一种标准化程度高、程序可移植性好的语言就是很重要的

⑦：软件的应用领域：所谓的通用程序设计语言实际上并不是对所有应用领域都同样适用。

三：编码风格

（1）程序内部的文档

A：定义

程序内部的文档包括恰当的标识符、适当的注解和程序的视觉组织

B：命名规则

① 选取含义鲜明的名字，使它能正确地提示程序对象所代表的实体

② 使用缩写时，缩写规则应该一致，并且应该给每个名字加注解

C：注释规则

① 在每个模块开始处应有序言性注解，简述功能、算法、接口、重要数据和开发简史

② 程序中间应有一段与代码有关的注解，主要解释代码的必要性

③ 对于用高级语言书写的源程序，应该利用注解提供一些额外的信息

④ 应该用空格或空行清楚地区分注解和程序

⑤ 注解的内容一定要正确，错误的注解会妨碍对程序的理解

D：程序清单布局规则

应利用适当的阶梯形式使程序的层次结构清晰明显

（2）数据说明

数据说明的次序应该标准化。有次序就容易查阅，因此能够加速测试、调试和维护的过程

多个变量名在一个语句中说明时，应该按字母顺序排列这些变量

使用复杂的数据结构，应用注解说明用设计语言实现这个数据结构的方法和特点

（3）语句构造

A：原则

每个语句都应该简单而直接，不能为了提高效率而使程序变得过分复杂

B：使语句简单明了的规则

① 不要为了节省空间而把多个语句写在同一行

② 尽量避免复杂的条件测试

③ 尽量减少对“非”条件的测试

④ 避免大量使用循环嵌套和条件嵌套

⑤ 利用括号使逻辑表达式或算术表达式的运算次序清晰直观

（4）输入输出

①对所有输入数据都进行检验

②检查输入项重要组合的合法性

③保持输入格式简单

④使用数据结束标记，不要要求用户指定数据的数目

⑤明确提示交互式输入的请求，详细说明可用的选择或边界数值

⑥当程序设计语言对格式有严格要求时，应保持输入格式一致

⑦设计良好的输出报表

⑧给所有输出数据加标志

（5）效率

A：程序运行时间

① 写程序之前先简化算术的和逻辑的表达式

② 仔细研究嵌套的循环，以确定是否有语句可以从内层往外移

③ 尽量避免使用多维数组

④ 尽量避免使用指针和复杂的表

⑤ 使用执行时间短的算术运算

⑥ 不要混合使用不同的数据类型

⑦ 尽量使用整数运算和布尔表达式

⑧ 在效率是决定性因素的应用领域，尽量使用有良好优化特性的编译程序，以自动生成高效目标代码

B：存储器效率

① 使用能保持功能域的结构化控制结构，是提高效率的好方法

② 在要求使用最少的存储单元时，应选用有紧缩存储器特性的编译程序

③ 提高执行效率的技术能提高存储器效率。提高存储器效率的关键是“简单”

C：输入输出效率

① 输入输出都应该有缓冲，以减少用于通信的额外开销

② 对二级存储器选用最简单的访问方法

③ 二级存储器的输入输出应以信息组为单位进行

④ 如果“超高效的”输入输出很难被人理解，则不应采用这种方法

第二节：软件测试基础

一：软件测试的目标（了解）

① 测试是为了发现程序中的错误而执行程序的过程

② 好的测试方案是极可能发现迄今为止尚未发现的错误的测试方案

③ 成功的测试是发现了至今为止尚未发现的错误的测试

④ 软件测试的根本目标是尽可能多地发现并排除软件中潜藏的错误，最终把一个高质量的软件系统交给用户使用

二：软件测试准则（了解）

① 所有测试都应该能追溯到用户需求

② 应该远在测试之前就制定测试计划

③ Pareto 原理：80%的错误是由 20%的模块造成的

④ 应该从小规模测试开始，并逐步进行大规模测试

⑤ 穷举测试是不可能的，测试只能证明程序有错误，而不能证明程序没有错误

⑥ 为了尽最大可能的发现错误，应该由独立的第三方担任测试工作

三：软件测试方法

（1）黑盒测试（功能测试）

把软件看成一个黑盒子，不考虑其内部结构和处理过程，只按照规格说明书的规定，测试软件是否能够正确接收输入数据，并产生正确的输出数据。也即测试程序是否正确实现了其功能

（2）白盒测试

把软件看作一个透明的盒子，完全知道程序内部结构和处理算法，根据程序内部的逻辑结构测试程序内部的主要执行通路是否能够按照预定的要求正确工作

四：软件测试步骤

（1）单元测试（模块测试）

单元测试是把每个模块作为一个单独的实体来测试，检验其正确性。目的在于保证每个模块作为一个单元能够正确运行

模块测试所发现的是编译和详细设计的错误

（2）子系统测试

子系统测试是把经过单元测试的模块放在一起形成一个子系统来测试。模块相互间协调和通信是此测试的主要问题，也即子系统测试着重测试模块的接口

（3）系统测试

系统测试是把经过测试的子系统装配成一个完整的系统来测试。在这个过程中不仅应该发现设计和编码的错误，还应该验证系统确实能够提供需求说明书中指定的功能，而且系统动态特性也符合预定要求

系统测试发现的往往是软件设计中的错误，也可能发现需求说明书中的错误

子系统测试和系统测试总称为为集成测试

（4）验收测试（确认测试）

验收测试是把软件系统作为单一的实体进行测试，它是在用户积极参与下进行的，而且主要使用实际数据进行测试，验收测试的目的是验证系统确实能够满足用户的需要

验收测试发现的是系统需求说明书中的错误

（5）平行运行

平行运行就是同时运行新开发出来的系统和将被它取代的旧系统，以便比较新旧两个系统的处理结果

第三节：单元测试

一：概述

单元测试集中检测软件设计的最小单元一模块，它和编码属于软件过程的同一个阶段。在编写出源程序代码并通过了编译程序的语法检查之后，就可以用详细设计描述作指南，对重要的执行通路进行测试，以便发现模块内部的错误。单元测试主要使用白盒测试技术，而且对多个模块的测试可以并行地进行，包括人工测试和计算机测试两种

测试依据：详细设计文档

测试技术：白盒测试技术

测试方法：人工测试和计算机测试

二：测试重点（了解）

（1）模块结构

模块接口的数据流是否能正常进出

参数的数目、次序、属性或单位系统与变元是否一致

是否修改了只作输入用的变元

全局变量的定义和用法在各个模块中是否一致

（2）局部数据结构

对于模块来说，局部数据结构是常见的错误来源。应该仔细设计测试方案，以便发现局部数据说明、初始化、默认值等方面的错误。

（3）重要的执行通路

选择最有代表性、最可能发现错误的执行通路进行测试

设计测试方案来发现由于错误计算、不正确的比较或不适当的控制流而造成的错误

（4）出错处理通路（了解）

当评价出错处理通路时，应着重测试可能发生的错误为:

① 对错误的描述是难以理解的

② 记下的错误与实际遇到的错误不同

③ 在对错误进行处理之前，错误条件已经引起系统干预

④ 对错误的处理不正确

⑤ 描述错误的信息不足以帮助确定造成错误的位置

（5）边界条件

边界测试是单元测试中最重要的任务。软件常常在它的边界上失效，例如，处理 n 元数组的第 n 个元素时，或做到 1 次循环中的第 1 次重复时，往往会发生错误。使用刚好小于、刚好等于和刚好大于最大值或最小值的数据结构、控制量和数据值的测试方案，非常可能发现软件中的错误。

三：测试方法

（1）代码审查

A：定义（了解）

人工测试源程序可以由程序的编写者本人非正式地进行，也可以由审查小组正式进行。后者称为代码审查是一种非常有效的程序验证技术。

B：流程

组建审查小组

审查会议

C：优点（了解）

一次审查会上可以发现许多错误

不需要每次发现一个错误就进行验证，减少了系统验证的总工作量

D：与计算机测试的关系

对于查找某些类型的错误来说，人工测试比计算机测试更有效;对于其他类型的错误来说则刚好相反。因此,人工测试和计算机测试是互相补充，相辅相成的，缺少其一都会使查找错误的效率降低。

（2）计算机测试（了解）

模块并不是一个独立的程序，因此必须要为每个单元测试开发驱动软件和(或)存根软件，具体如下:

驱动程序：接收测试数据，把这些数据传送给被测试的模块，并且印出有关的结果

存根程序：代替被测试的模块所调用的模块,它使用被它代替的模块的接口，做最少量的数据操作，印出对入口的检验或操作结果，并且把控制归还给调用它的模块

第四节：集成测试

一：概念

（1）方法分类

非渐增测试： 先分别测试每个模块，再把所有模块按设计要求放在一起结合成所要的程序

渐增测试： 把下一个要测试的模块同已经测试好的那些模块结合起来进行测试,测试完以后再把下一个应该测试的模块结合进来测试，每次增加一个模块。渐增式测试同时完成单元测试和集成测试

（2）非渐增测试的缺点（了解）

把所有模块放在一起，测试者面对的情况十分复杂

在庞大的程序中诊断定位一个错误非常困难

一旦改正一个错误之后，又会遇到新的错误，没有穷尽

（3）渐增测试的优点（了解）

把程序划分成小段来构造和测试，比较容易定位和改正错误

对接口可以进行更彻底的测试

可以使用系统化的测试方法

二：渐增式测试策略

（1）自顶向下集成

A：定义

从主控制模块开始，沿着程序的控制层次向下移动，逐渐把各个模块结合起来。在把附属于主控制模块的模块组装到程序结构中时，使用深度优先的策略或宽度优先的策略

B：步骤（了解）

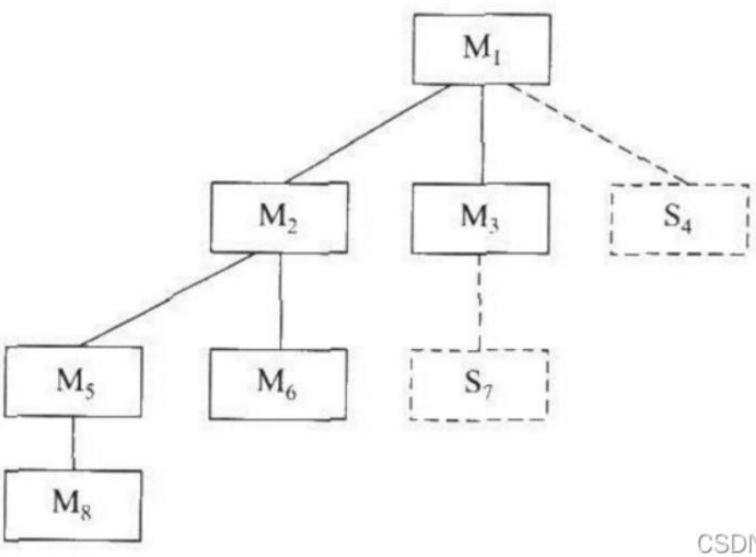
对主控制模块进行测试，测试时用存根程序代替所有直接附属于主控制模块的模块

根据选的结合策略(深度优先或宽度优先)，每次用一个实际模块代换一个存根程序

在结合进一个模块的同时进行测试

为了保证加入模块没有引进新的错误，可能需要进行回归测试

C：结合策略（了解）



CSDN @快乐

①：深度优先

深度优先的结合方法先组装在软件结构的一条主控制通路上的所有模块。步骤如下:

- 第一，选择一条主控制通路取决于应用的特点(如，选取左通路)
- 第二，结合模块M_1、M_2、M_5
- 第三，把M_8或M_6结合进来
- 第四，构造中央的和右侧的控制通路

②：宽度优先

宽度优先的结合方法是沿软件结构水平地移动,把处于同一个控制层次上的所有模块组装起来。步骤如下

- 第一，结合模块M_2、M_3、M_4
- 第二，结合下一个控制层次中的模块M_5、M_6、M_7
- 第三，继续进行下去，直到所有模块都被结合进来为止

D：优缺点

优点

① 不需要测试驱动程序
② 能够在测试阶段的早期实现并验证系统的主要功能
③ 能在早期发现上层模块的接口错误

缺点

① 需要存根程序，可能遇到与此相联系的测试困难
② 低层关键模块中的错误发现较晚
③ 在早期不能充分展开人力

（2）自底向上集成

A：定义

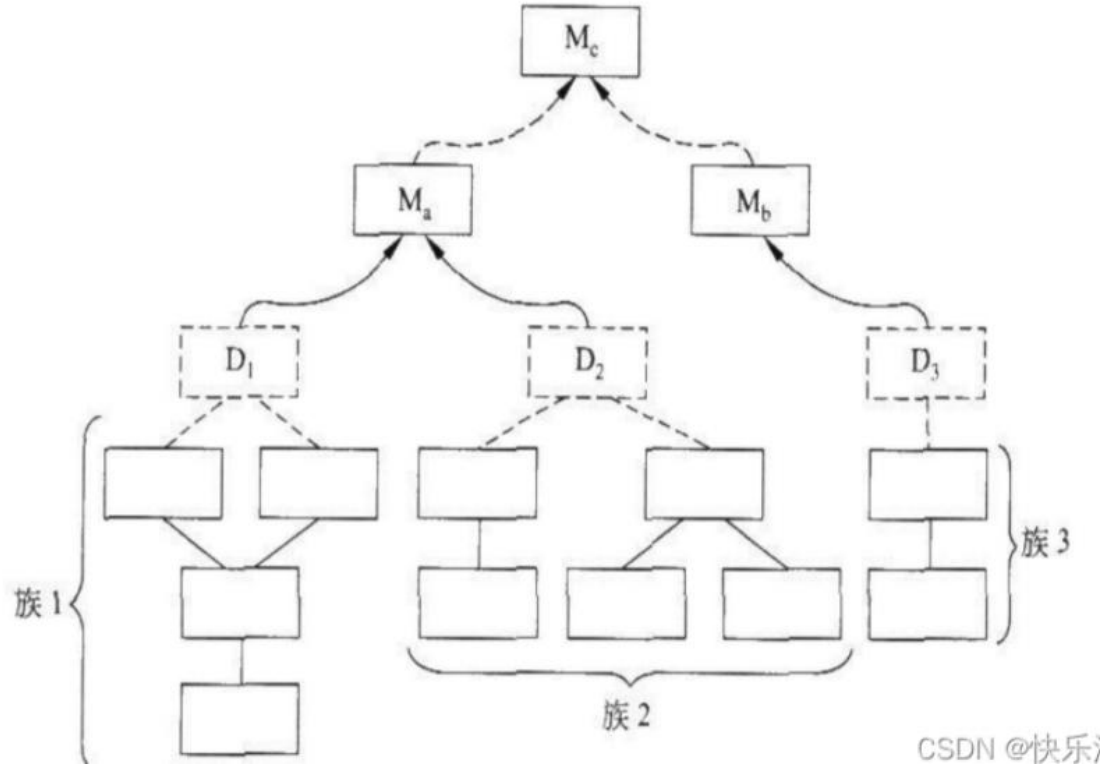
自底向上测试从软件结构最低层的模块开始组装和测试。因为是从底部向上结合模块，总能得到所需的下层模块处理功能，所以不需要存根程序

B：步骤（了解）

① 把低层模块组合成实现某个特定的软件子功能的族
② 写一个用于测试的控制程序，协调测试数据的输入和输出
③ 对由模块组成的子功能族进行测试
④ 去掉驱动程序，沿软件结构自下向上移动，把子功能族组合起来形成大的子功能族

C：案例

- 把模块组合成族1、族2和族3
- 使用驱动程序（图中虚线方框）对每个子功能族进行测试
- 族1和族2中的模块属于模块M_a，去掉驱动程序D_1和D_2，把这两个族直接同M_a连接起来。同样在和模块M_b结合之前去掉族3驱动程序D_3
- M_a和M_b这两个模块都与模块M_c结合起来



CSDN @快乐

D：优缺点

优点

① 不需要存根程序，不会遇到与此相联系的测试困难
② 能较早发现低层关键模块中的错误
③ 在早期能充分展开人力

缺点

① 需要测试驱动程序
② 不能够在测试阶段的早期实现并验证系统的主要功能
③ 不能在早期发现上层模块的接口错误

三：回归测试

（1）定义

回归测试是指重新执行已经做过的测试的某个子集，以保证上述这些变化没有带来非预期的副作用。它可以用于保证由于调试或其他原因引起的变化，不会导致非预期的软件行为或额外错误的测试活动

（2）方法（了解）

通过重新执行全部测试用例的一个子集人工地进行

利用捕获回放工具，捕获测试用例和实际运行结果，然后回放，并比较运行结果

（3）回归测试集（了解）

回归测试集(已执行过的测试用例的子集)包括下述 3 类不同的测试用例:

① 检测软件全部功能的代表性测试用例
② 专门针对可能受修改影响的软件功能的附加测试
③ 针对被修改过的软件成分的测试

第五节：确认测试

一：概念

确认测试也称为验收测试，它的目标是验证软件的有效性

验证： 为了保证软件正确的实现某个特定要求而进行的一系列活动

确认：为了保证软件确实满足了用户需求而进行的一系列活动

软件有效性：如果软件的功能和性能如同用户所合理期待的那样，软件就是有效的

二：确认测试的范围（了解）

（1）要求

必须有用户积极参与，或者以用户为主进行。用户应该参与设计测试方案，使用用户界面输入测试数据并且分析评价测试的输出结果。在验收之前由开发单位对用户进行培训

确认测试通常使用黑盒测试法。应该仔细设计测试计划和测试过程，测试计划包括要进行的测试的种类及进度安排，测试过程规定了用来检测软件是否与需求一致的测试方案。通过测试和调试要保证软件能满足所有功能要

（2）结果

功能和性能与用户要求一致，软件是可以接受的

功能和性能与用户要求有差距

三：软件配置复查（了解）

（1）目的

保证软件配置的所有成分都齐全，质量符合要求，文档与程序完全一致， 具有完成软件维护所必须的细节，而且已经编好目录

（2）要求

在确认测试过程中应该严格遵循用户指南及其他操作程序

必须仔细记录发现的遗漏或错误，并且适当地补充和改正

四：Alpha 测试 Beta 测试

（1）Alpha 测试

Alpha 测试由用户在开发者的场所进行，并且在开发者对用户的“指导”下进行测试，且开发者负责记录发现的错误和遇到的问题。即 Alpha 测试是在受控的环境中进行的

（2）Beta 测试

Beta 测试由软件的最终用户们在一个或多个客户场所进行。开发者通常不在 Beta 测试的现场，即 Beta 测试是软件在开发者不能控制的环境中的“真实”应用

第六节：白盒测试

一：逻辑覆盖

（1）定义

逻辑覆盖是对一系列测试过程的总称， 这组测试过程逐渐进行越来越完整的通路测试

（2）分类

① 语句覆盖：选择足够多的测试数据，被测试程序中的每条语句至少执行一次

② 判定覆盖：不仅每个语句至少执行一次，而且每个判定的每种可能的结果都应该至少执行一次

③ 条件覆盖：不仅每个语句至少执行一次，而且使判定表达式中的每个条件都取到各种可能的结果

④ 判定/条件覆盖：选择足够多的测试数据，使判定表达式中的每个条件都取到各种可能的结果，而且每个判定表达式也都取到各种可能的结果。它同时满足判断覆盖和条件覆盖

⑤ 条件组合覆盖：选取足够多的测试数据，使得每个判定表达式中条件的各种可能组合都至少出现一次。满足条件组合覆盖，也一定满足判定覆盖、条件覆盖和判断/条件覆盖

⑥ 点覆盖：连通图 G 的子图 G’ 是连通的，而且包含 G 的所有结点，则称 G’ 是 G 的点覆盖。满足点覆盖标准要求选取足够多的测试数据，使得程序执行路径至少经过流图的每个结点一次，也即点覆盖标准和语句覆盖标准是相同的

⑦ 边覆盖：连通图 G 的子图 G’’ 是连通的，而且包含 G 的所有边，则称 G’’ 是 G 的边覆盖。为满足边覆盖的测试标准，要求选取足够多的测试数据，使程序执行路径至少经过流图每条边一次，也即边覆盖与判定覆盖是相同的

⑧ 路径覆盖：选取足够多的测试数据，使程序的每条可能路径都至少执行一次，如果程序图中有环，则要求每个环至少经过一次

二：控制结果测试

（1）基本路径测试

A：定义（了解）

基本路径测试是 T o m M c C a b e TomMcCabeTomMcCabe 提出的一种白盒测试技术。使用这种技术设计测试用例时，首先计算程序的环形复杂度，并用该复杂度为指南，定义执行路径的基本集合,从该基本集合导出的测试用例可以保证程序中的每条语句至少执行一次，而且每个条件在执行时都将分别取真、假两种值

B：步骤（了解）

① 根据过程设计结果画出相应的流图

② 计算流图的环形复杂度

③ 确定线性独立路径(至少包含一条在定义该路径之前不曾用过的边)的基本集合

④ 设计可强制执行基本集合中每条路径的测试用例

（2）条件测试（了解）

A：关系表达式

一个简单条件是一个布尔变量或一个关系表达式，在布尔变量或关系表达式之前还可能有一个 NOT(¬ \neg¬)运算符，关系表达式的形式如下：

$$E_1 < \text{关系算符} > E_2$$

其中E_1和E_2是**算数表达式**，而<关系算符>是下列算符之一

- $<$
- $\leq$
- $=$
- $\neq$
- $>$
- $\geq$

2

复合条件由两个或多个简单条件、布尔算符和括弧组成

布尔算符有：

- OR（|）
- AND（&）
- NOT(¬)

不包含关系表达式的条件称为布尔表达式

B：条件错误的类型

① 布尔算符错;
② 布尔变量错;
③ 布尔括弧错;
④ 关系算符错;
⑤ 算术表达式错

C：条件测试的优点

① 容易度量条件的测试覆盖率
② 程序内条件的测试覆盖率可指导附加测试的设计

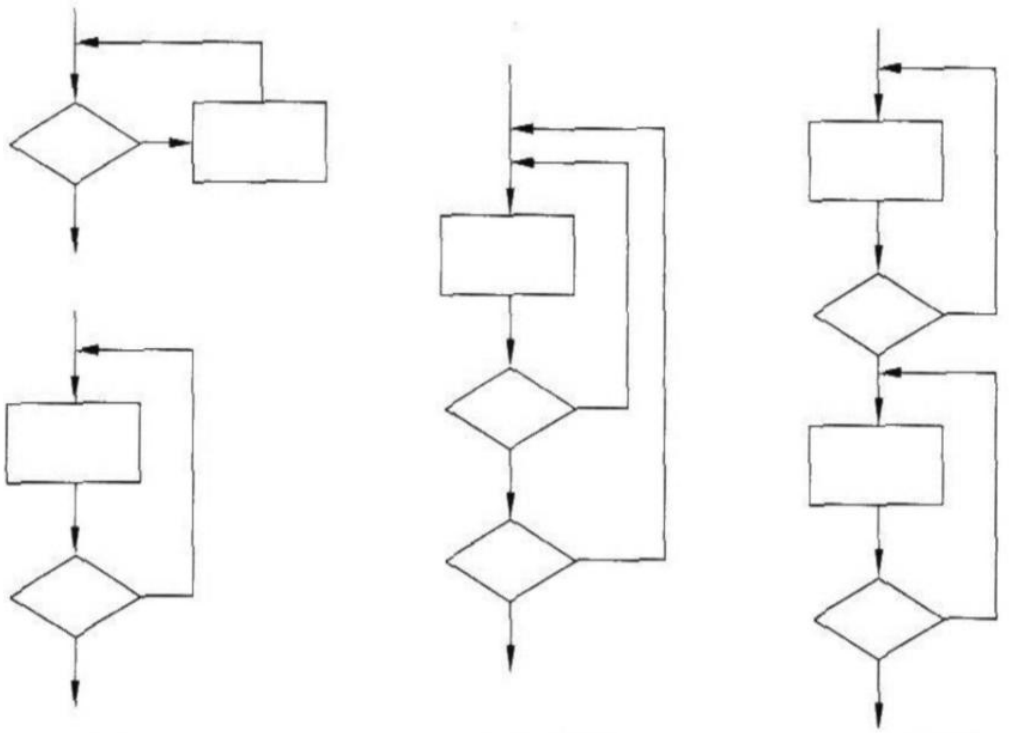
（3）循环测试（了解）

A：定义

循环是绝大多数软件算法的基础，但是,在测试软件时却往往未对循环结构进行足够的测试。循环测试是一种白盒测试技术，它专注于测试循环结构的有效性

B：分类

在结构化的程序中通常只有 3 种循环，即简单循环、串接循环和嵌套循环



(a) 简单循环　　(b) 嵌套循环　　(c) 串接循环

第七节：黑盒测试

一：概念

（1）目的

① 功能不正确或遗漏了功能

② 界面错误

③ 数据结构错误或外部数据库访问错误

④ 性能错误

⑤ 初始化和终止错误

（2）适用性

白盒测试在测试过程的早期阶段进行，黑盒测试主要用于测试过程的后期

（3）设计测试方案时需要考虑的问题（了解）

① 怎样测试功能的有效性?

② 哪些类型的输入可构成好测试用例?

③ 系统是否对特定的输入值特别敏感?

④ 怎样划定数据类的边界?

⑤ 系统能够承受什么样的数据率和数据量?

⑥ 数据的特定组合将对系统运行产生什么影响?

（4）测试用例的标准

① 能够减少为达到合理测试所需要设计的测试用例的总数

② 能够告诉人们，是否存在某些类型的错误，而不是仅仅指出与特定测试相关的错误是否存在

二：技术方法

（1）等价类划分法

A：定义

这种技术把程序的输入域划分成若干个数据类，据此导出测试用例，一个理想的测试用例能独自发现一类错误

B：目的

等价划分法力图设计出能发现若干类程序错误的测试用例，从而减少必须设计的测试用例的数目

C：流程

①：划分数据的等价类

第一，需要研究程序的功能说明，从而确定输入数据的有效等价类和无效等价类

第二，在确定输入数据的等价类时常常还需要分析输出数据的等价类

第三，在划分等价类时还应考虑编译程序的检错功能

②：根据等价类设计测试方案

第一，设计一个新的测试方案以尽可能多地覆盖尚未被覆盖的有效等价类，重复这一步骤直到所有有效等价类都被覆盖为止

第二，设计一个新的测试方案，使它覆盖一个而且只覆盖一个尚未被覆盖的无效等价类，重复这一步骤直到所有无效等价类都被覆盖为止

D：规则（了解）

① 如果规定了输入值的范围，则可划分一个有效的等价类(输入值在此范围内)，两个无效的等价类(输入小于最小值或大于最大值)

② 如果规定了输入数据的个数，则类似地也可以划分出一个有效的等价类和两个无效的等价类

③ 如果规定了输入数据的一组值，而且程序对不同输入值做不同处理，则每个允许的输入值是一个有效的等价类，此外还有一个无效的等价类(任一个不允许的输入值)

④ 如果规定了输入数据必须遵循的规则，则可以划分出一个有效的等价类(符合规则)和若干个无效的等价类（从各种不同角度违反规则)

⑤ 如果规定了输入数据为整型，则可以划分出正整数、零和负整数 3 个有效类

⑥ 如果程序的处理对象是表格，则应该使用空表，以及含一项或多项的表

（2）边界值分析法

使用边界值分析方法设计测试方案首先应该确定边界情况，选取的数据应该刚好等于、稍小于和稍大于等价类边界值，即应该选取刚好等于、稍小于和稍大于等价类边界值的数据作为测试数据，而不是选取每个等价类内的典型值或任意值作为测试数据

（3）错误推测（了解）

错误推测法基本思想是列举出程序中可能有的错误和容易发生错误的特殊情况，并且根据它们选择测试方案

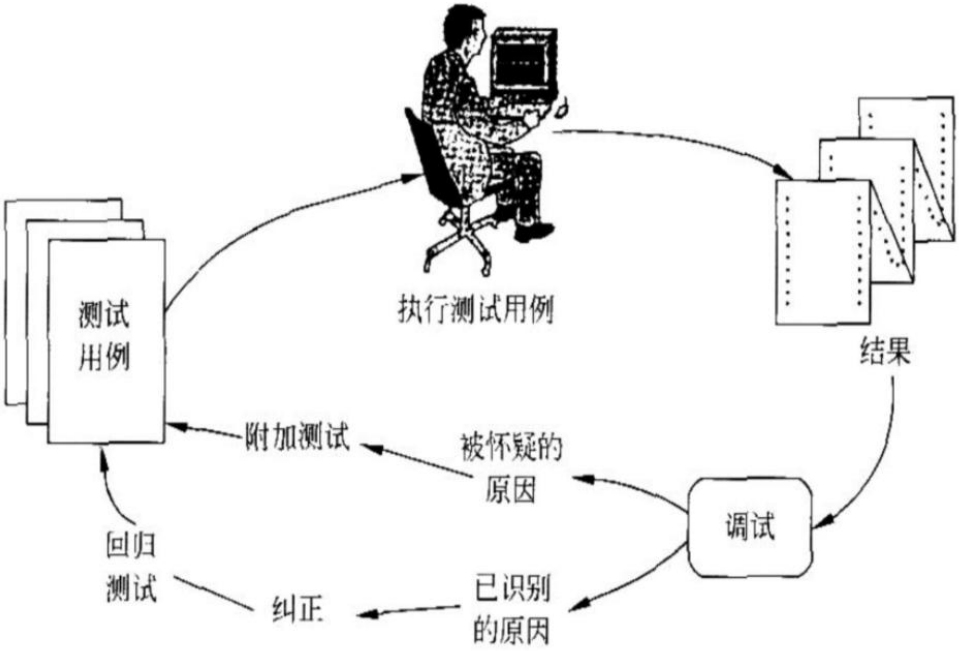
第八节：调试

一：定义

调试是在测试发现错误之后排除错误的过程

二：调试流程（了解）

（1）流程



（2）软件错误的特征

① 症状和产生症状的原因可能在程序中相距甚远

② 当改正了另一个错误之后，症状可能暂时消失了

③ 症状可能实际上并不是由错误引起的

④ 症状可能是由不易跟踪的人为错误引起的

⑤ 症状可能是由定时问题而不是由处理问题引起的

⑥ 可能很难重新产生完全一样的输入条件

⑦ 症状可能时有时无

⑧ 症状可能是由分布在许多任务中的原因引起的，这些任务运行在不同的处理机上

三：调试途径

（1）蛮干法

A：思路

按照“让计算机自己寻找错误”的策略，这种方法印出内存的内容，激活对运行过程的跟踪，并在程序中到处都写上 WRITE (输出)语句，在生成的信息海洋的某个地方发现错误原因的线索

B：适用性

蛮干法是寻找软件错误原因的最低效的方法。仅当所有其他方法都失败了的情况下，才应该使用这种方法

（2）回溯法

A：思路

从发现症状的地方开始，人工沿程序的控制流往回追踪分析源程序代码，直到找出错误原因为止

B：适用性

当调试小程序时回溯法非常有效的。但随着程序规模的扩大,应该回溯的

路径数目也变得越来越大，以至彻底回溯变成完全不可能了

（3）原因排除法

对分查找法：

归纳法

演绎法

四： 调试准则（了解）

仔细分析程序出错处的逻辑模式，找出该错误出现的所有地方

在改正错误前应仔细研究源程序，以评估逻辑和数据结构的耦合程度

修改软件产品的同时改进开发软件产品的软件过程，避免今后在程序中出现错误

第七章：软件维护

第一节：软件维护的概念和特点

一：软件维护的概念

（1）定义

软件维护是在软件已经交付使用后，为了改正错误或满足新的需要而修改软件的过程，是软件生命周期的最后一个阶段，其基本任务是保证软件在一个相当长的时期能够正常运行

（2）分类

① 改正性维护：诊断和改正错误的过程（17%~21%）

② 适应性维护：为了和变化了的环境适当地配合而进行的修改软件的活动（18%~25%）

③ 完善性维护：为了满足用户提出的增加新功能或修改已有功能的要求和一般性改进要求（50%~66%）

④ 预防性维护：当为了改进未来的可维护性或可靠性，或为了给未来的改进奠定更好的基础而修改软件（4%）

二：软件维护的特点

（1）结构化维护和非结构化维护差别巨大

非结构化维护：唯一成分是程序代码，维护活动从艰苦地评价程序代码开始，需要付出很大代价

结构化维护：有完整的软件配置存在，维护工作从评价设计文档开始

（2）维护的代价高昂（了解）

- 因为可用的资源必须供维护任务使用，以致耽误甚至丧失了开发的良机
- 当看来合理的有关改错或修改的要求不能及时满足时将引起用户不满
- 由于维护时的改动，在软件中引入了潜伏的错误，从而降低了软件的质量
- 当必须把软件工程师调去从事维护工作时，将在开发过程中造成混乱
- 生产率的大幅度下降

（3）维护存在很多问题（了解）

- 理解别人写的程序非常困难
- 维护的软件往往没有合格的文档，或者文档材料显著不足
- 要求对软件进行维护时，不能指望由开发人员给我们仔细说明软件
- 绝大多数软件在设计时没有考虑将来的修改
- 软件维护不是一项吸引人的工作

第三节：软件可维护性

一：定义

可维护性指的是维护人员理解、改正、改动或改进这个软件的难易程度。提高可维护性是支配软件工程方法学所有步骤的关键目标

二：决定软件可维护性的因素

（1）可理解性

A：定义

软件可理解性表现为外来读者理解软件的结构、功能、接口和内部处理过程的难易程度

B：影响因素

① 模块化(模块结构良好，高内聚，松耦合);

② 详细的设计文档;

③ 结构化设计;

④ 程序内部的文档;

⑤ 高级程序设计语言等

（2）可测试性

诊断和测试的容易程度取决于软件容易理解的程度

A：影响因素

① 良好的文档;

② 软件结构;

③ 可用的测试工具和调试工具;

④ 以前设计的测试过程

B：要求

维护人员需要得到在开发阶段用过的测试方案，以便进行回归测试。在设计阶段应该尽力把软件设计成容易测试和容易诊断的

C：衡量标准

对于程序模块来说，可以用程序复杂度来度量它的可测试性。模块的环形复杂度越大，可执行的路径就越多，全面测试它的难度就越高

（3）可修改性

耦合、内聚、信息隐藏、局部化、控制域与作用域的关系等，都影响软件的可修改性

（4）可移植性

A：定义

软件可移植性是指把程序从一种计算环境(硬件配置和操作系统)转移到另一种计算环境的难易程度

B：提高可移植性的方法

把与硬件、操作系统以及其他外部设备有关的程序代码集中放到特定的程序模块中，可以把因环境变化而必须修改的程序局限在少数程序模块中，从而降低修改的难度，提高可移植性

（5）可重用性

A：定义

重用是指同一事物不做修改或稍加改动就在不同环境中多次重复使用

B：对可维护性的影响

① 提高软件可靠性，较少改正性维护

② 降低适应性和完善性维护的难度

第八章：面向对象方法学

第一节：面向对象方法学

一：要点

（1）基本原则

面向对象方法学的出发点和基本原则，是尽可能模拟人类习惯的思维方式，使开发软件的方法与过程尽可能接近人类认识世界解决问题的方法与过程，使描述问题的问题空间(问题域)与实现解法的解空间(求解域)在结构上尽可能一致

（2）定义

面向对象方法是一种以数据或信息为主线，把数据和处理相结合的方法，即把对象作为由数据及可以施加在这些数据上的操作所构成的统一体。面向对象的方法可以用下列方程来概括:

OO = object（对象） + classes（类） + inheritance（继承） + communication with messages（封装）

（3）要点

对象：面向对象的软件系统是由对象组成的，软件中的任何元素都是对象，复杂的软件对象由比较简单的对象组合而成。用对象分解取代了传统方法的功能分解，对象是从客观世界中的实体抽象而来的，是不固定的

类：把所有对象都划分成各种对象类,每个对象类都定义了一组数据和一组方法。数据用于表示对象的静态属性,是对象的状态信息。类中定义的方法，是允许施加于该类对象上的操作，是该类所有对象共享的，并不需要为每个对象都复制操作的代码

在 C++中，类定义如下，其中成员函数（也即方法）类只保留一份，实例化后的对象在调用时隐含传入 this 指针

```
struct Student
{
	char _name[20];//数据
	char _gender[3];
	int _age;

	//方法
	void SetStudentInfo(const char* name,const char* gender,int age)
	{
		strcpy(_name,name);
		strcpy(_gender,gender);
		_age=age;
	}
};

int main()
{
	Student s;
	s.SetStudentInfo("Bob","男",18);
```

```
    return 0;
}
```

继承性：按子类与父类的关系，把若干个对象类组成一个层次结构的系统。子类自动具有和上层的父类相同的数据和方法，而且低层的特性将屏蔽高层的同名特性

① 在 C++中，子类可以继承父类，有 public，private，protect 三种继承方式

② 子类成员会对同名父类成员进行屏蔽，这种情况我们称之为隐藏，或者叫做重定义，具体可见 8-3：C++继承之继承中的作用域，隐藏，重定义和静态成员

封装性：对象彼此之间仅能通过传递消息互相联系。对象是进行处理的主体，必须发消息请求它执行它的某个操作，处理它的私有数据，而不能从外界直接对它的私有数据进行操作。一切局部于该对象的私有信息，都被封装在该对象类的定义中，就好像装在一个不透明的黑盒子中一样，在外界是看不见的，更不能直接使用，

在 C++中可以体会到，类就是如此，它对外体现的是一个整体，不暴露自己的细节，只提供接口

二：优点

1.与人类习惯的思维方法一致

① 以对象为核心，开发出的软件系统由对象组成

② 设计的主要思路是使用现实世界的概念抽象地思考问题从而自然地解决问题

③ 基本原则是按照人类习惯的思维方法建立问题域和求解域模型

④ 抽象机制使用户在利用计算机软件系统解决复杂问题时使用习惯的抽象思维工具

⑤ 对象分类过程，支持从特殊到一般的归纳思维过程

⑥ 继承特性，支持从一般到特殊的演绎思维过程

⑦ 提供了随着对系统认识的逐步深入和具体化，而逐步设计和实现该系统的可能性

2.稳定性好

面向对象方法基于构造问题领域的对象模型，以对象为中心构造软件系统。它的基本作法是用对象模拟问题领域中的实体，以对象间的联系刻画实体间的联系。因为面向对象的软件系统的结构是根据问题领域的模型建立起来的，而不是基于对系统应完成的功能的分解，当对系统的功能需求变化时不会引起软件结构的整体变化，仅需要作一些局部性的修改

3.可重用性

重用技术是用已有的零部件装配新的产品。重用是提高生产率的最主要的方法。对象是比较理想的模块和可重用的软件成分

4.较易开发大型软件产品

面向对象方法学开发软件时，把一个大型软件产品分解成一系列本质上相互独立的小产品来处理，不仅降低了开发的技术难度，而且使开发工作的管理变容易了

5.可维护性好

这是因为

① 面向对象的软件稳定性比较好

② 面向对象的软件比较容易修改

③ 面向对象的软件比较容易理解

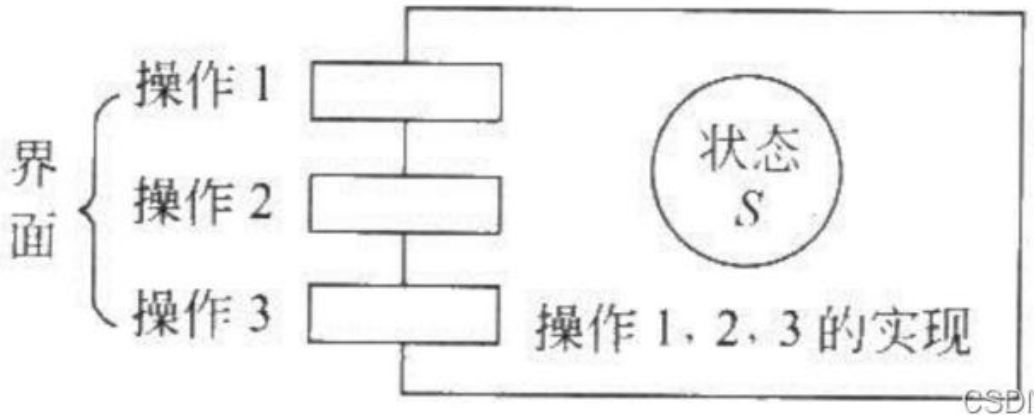
④ 易于测试和调试

第二节：面向对象的概念

一：对象（Object）

（1）对象的形象表示

下图形象地描绘了具有 3 个操作的对象。图 9-1 形象地描绘了具有 3 个操作的对象。面向对象方法学中的对象是由描述该对象属性的数据以及可以对这些数据施加的所有操作封装在一起构成的统一体。对象可以作的操作表示它的动态行为

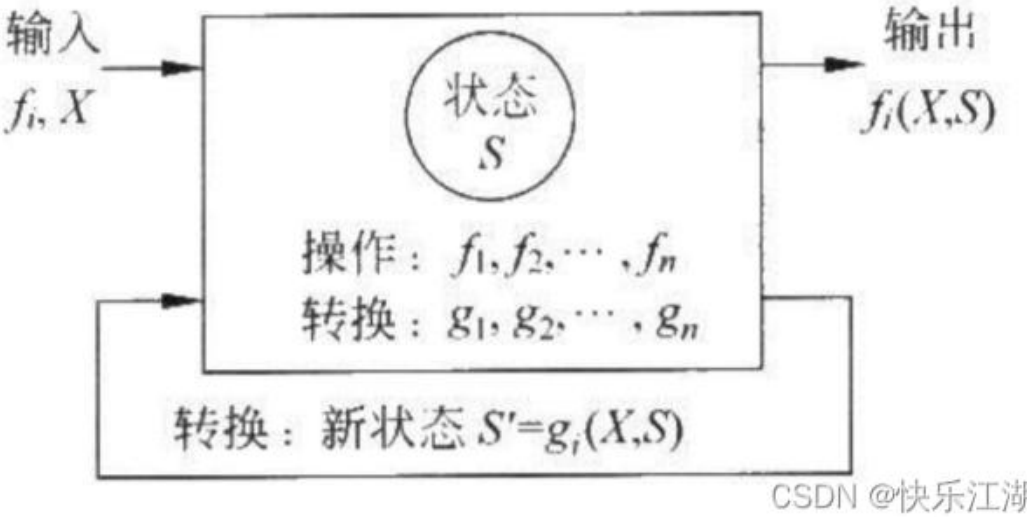


（2）对象的定义

按照面向对象程序设计的角度：对象是具有相同状态的一组操作的集合。 在应用领域中有意义的、与所要解决的问题有关系的任何事物都可以作为对象，它既可以是具体的物理实体的抽象，也可以是人为的概念，或者是任何有明确边界和意义的东西

按照结构化角度：对象是封装了数据结构及可以施加在这些数据结构上的操作的封装体,这个封装体有可以唯一地标识它的名字，而且向外界提供一组服务。对象中的数据表示对象的状态，一个对象的状态只能由该对象的操作来改变。每当需要改变对象的状态时，只能由其他对象向该对象发送消息。对象响应消息时，按照消息模式找出与之匹配的方法，并执行该方法

按照动态（对象的实现机制）角度：对象是一台自动机，具有内部状态 S



CSDN @快乐江湖

（3）对象的特点

1.以数据为中心

操作围绕对其数据所需要做的处理来设置，不设置与这些数据无关的操作，而且操作的结果往往与当时所处的状态有关

2.对象是主动的

对象进行处理的主体。不能从外部直接加工它的私有数据，而是必须通过它的公有接口向对象发消息，请求它执行它的某个操作，处理它的私有数据

3.实现了数据封装

私有的数据完全被封装在内部，对外是隐藏的、不可见的，对私有数据的访问或处理只能通过公有的操作进行。为了使用对象内部的私有数据，只需知道数据的取值范围和可以对该数据施加的操作。一个对象类型也可以看作是一种抽象数据类型。

4.具有并行性

对象是描述其内部状态的数据及可以对这些数据施加的全部操作的集合。不同对象各自独立地处理自身的数据，彼此通过发消息传递信息完成通信。因此，本质上具有并行工作的属性。

5.模块独立性好

对象是由数据及可以对这些数据施加的操作所组成的统一体，而且对象是以数据为中心的，操作围绕对其数据所需做的处理来设置，没有无关的操作。因此，对象内部各种元素彼此结合得很紧密，内聚性相当强。它与外界的联系比较少，对象之间的耦合比较松

二：类

“类”是对具有相同数据和相同操作的一组相似对象的定义,即类是对具有相同属性和行为的一个或多个对象的描述，包括对怎样创建该类的新对象的说明。类是支持继承的抽象数据类型，而对象就是类的实例

类就是在定义一种类型，而对象是类实例化出来的。类相当于一个模板，对象是根据模板造出来的

```
class className
{
    //成员函数
    //成员变量

};//注意分号
```

三：实例

实例就是由某个特定的类所描述的一个具体的对象。类是对具有相同属性和行为的一组相似的对象的抽象，类在现实世界中并不能真正存在。实际上类是建立对象时使用的“样板”，按照这个样板所建立的一个个具体的对象，就是类的实际例子，通常称为实例

```
ClassName object1；
ClassName object1；
....
```

四：消息

消息就是要求某个对象执行在定义它的那个类中所定义的某个操作的规格说明。一

个消息由接收消息的对象、消息选择符、零个或多个变元组成。

例如一个类中定义了某个打印函数，它要接受一个字符串，所以对象在执行时就会调用它，接着传入参数即可，这里的参数就是规格说明

五：方法

方法就是对象所能执行的操作，也就是类中所定义的服务。方法描述了对象执行操作的算法，响应消息的方法

例如 C++中类的成员函数就是方法

六：属性

属性就是类中所定义的数据，它是对客观世界实体所具有的性质的抽象。类的每个实例都有自己特有的属性值

七：封装

（1）定义

封装是把数据和实现操作的代码集中起来放在对象内部。封装也就是信息隐藏，通过封装对外界隐藏了对象的实现细节

（2）特点

① 有一个清晰的边界

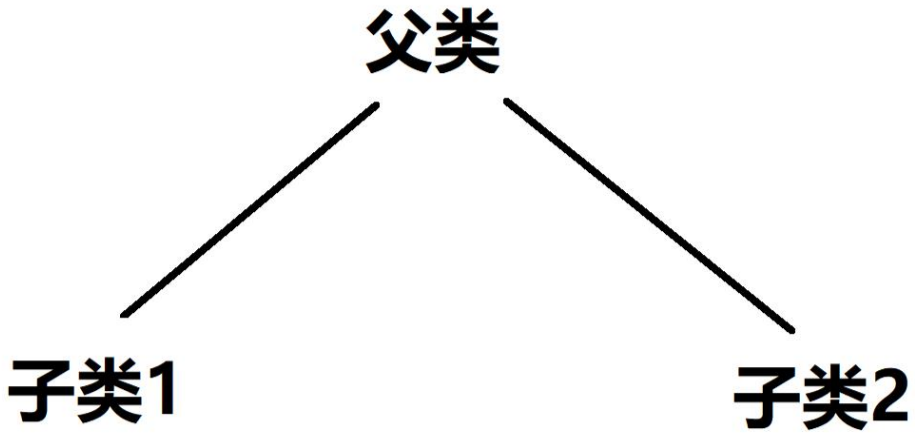
② 有确定的接口

③ 受保护的内部实现

八：继承

（1）定义

广义地说，继承是指能够直接获得已有的性质和特征，而不必重复定义它们。在面向对象的软件技术中，继承是子类自动地共享父类中定义的数据和方法的机制



（2）特点

① 继承具有传递性

② 子类将会屏蔽父类的同名性质

（3）继承的优点（了解）

① 使得相似的对象可以共享程序代码和数据结构，大大减少程序中的冗余信息

② 便于软件维护和开发

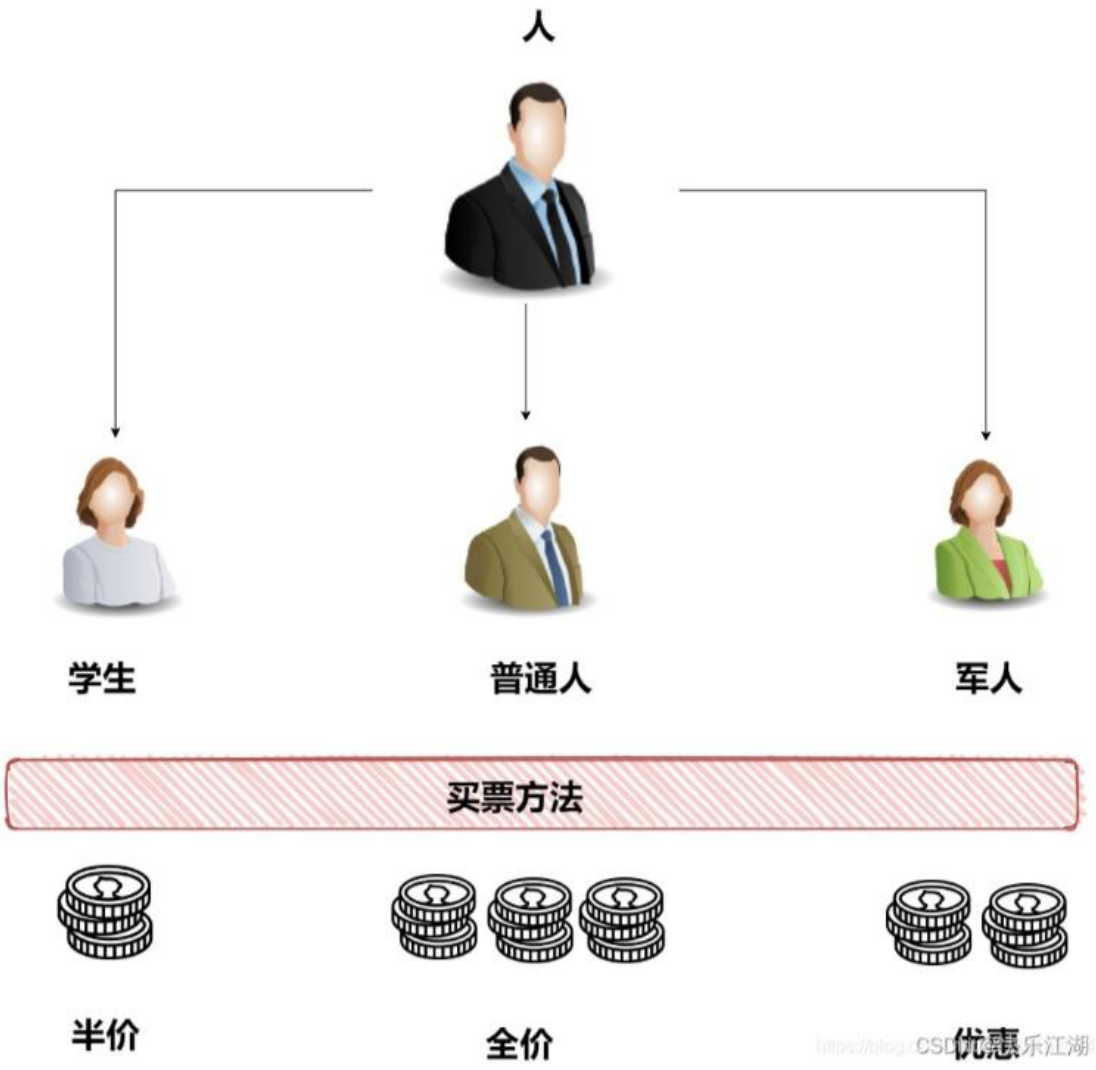
③ 用户在开发新系统时不需要从零开始，可以继承原有相似功能或者从类库中选取需要的类，然后再派生新类

九：多态性

（1）多态

多态性是指子类对象可以像父类对象那样使用，同样的消息既可以发送给父类对象也可以发送给子类对象。即在类等级的不同层次中可以共享一个方法的名字，不同层次中的每个类各自按自己的需要来实现这个行为

比如经典的买票行为，学生和普通人都属于人，但是学生买票可以半价，而普通人则只能全票，但是他们买票的这个接口实现的原理肯定是一样的，只是策略不同



C++实现如下

```
#include <iostream>
using namespace std;


class Person
{
public:
	void BuyTicket()
	{
		cout << "全价" << endl;
```

```
    }
};
class Student : public Person
{
public:
    void BuyTicket()
    {
        cout << "半价" << endl;
    }

};

void fun(Person& p)
{
    p.BuyTicket();
}

int main()
{
    Person Job;
    fun(Job);

    Student Bob;
    fun(Bob);
}
```

（2）优点

① 增加了面向对象软件系统的灵活性，进一步减少信息冗余

② 提高了可重用性和可扩充性

十：重载

函数重载：在同一作用域，若干个参数特征不同的函数可以使用相同的函数名

C++可以有同名函数的原因就是支持函数重载

运算符重载：同一个运算符可以施加于不同类型的操作数上面

第三节：面向对象建模之对象模型

一：概念

（1）定义

对象模型表示静态的、结构化的系统的数据性质。它是对模拟客观世界实体的对象以及对象彼此间的关系的映射，描述了系统的静态结构。对象模型为建立动态模型和功能模型，提供了实质性的框架。

（2）工具

使用 UML（统一建模语言）提供的类图来建立对象模型。在 UML 中，类的实际含义是一个类及属于该类的对象

具体来说，UML 提供了以下 13 种图

① 用例图：从用户角度描述系统功能。

② 类图：描述系统中类的静态结构。

③ 对象图：系统中的多个对象在某一时刻的状态。

④ 状态图：是描述状态到状态控制流，常用于动态特性建模

⑤ 活动图：描述了业务实现用例的工作流程

⑥ 顺序图：对象之间的动态合作关系，强调对象发送消息的顺序，同时显示对象之间的交互

⑦ 协作图：描述对象之间的协助关系

⑧ 构件图：一种特殊的 UML 图来描述系统的静态实现视图

⑨ 部署图：定义系统中软硬件的物理体系结构

⑩ 包图：对构成系统的模型元素进行分组整理的图

⑪ 组合结构图：表示类或者构建内部结构的图

⑫ 交互概览图：用活动图来表示多个交互之间的控制关系的图

二：类图的基本符号

（1）定义类

A：表示

UML 中类的图形符号为长方形，用两条横线把长方形分上、中、下 3 个区域，3 个区域分别放类的名字、属性和服务

B：命名规则

类名应该是富于描述的、简洁的而且无二义性的

① 使用标准术语，不要随意创造名字

② 使用具有确切含义的名词，不要使用空洞或含义模糊的词作名字

③ 必要时可用名词短语作名字，有时也可以加入形容词

（2）定义属性

具体格式为

可见性　属性名：类型名=初值{性质串}

可见性：有公有的（+）、私有的（-）和保护的（#）

类型名：表示该属性的数据类型

赋值：在创建类的实例时应给其他属性赋值，如果给某个属性定义了初值，则该初值可作为创建实例时这个属性的默认值

性质串：明确地列出该属性所有可能取值，用逗号隔开

（3）定义服务

具体格式为

可见性　操作名（参数表）：返回值类型{性质串}

可见性：有公有的（+）、私有的（-）和保护的（#）

参数表：用逗号隔开不同参数，每个参数语法为 “参数名:类型名=默认值”

三：表示关系的符号

类与类之间通常具有以下四种关系

（1）关联

A：定义

关联表示两个类的对象之间存在某种语义上的联系

B：关联的角色

在任何关联中都会涉及参与此关联的对象所扮演的角色，在某些情况下显式标明角色名有助于别人理解类

如果没有显式标出角色名，则意味着用类名作为角色名

C：普通关联

①：定义

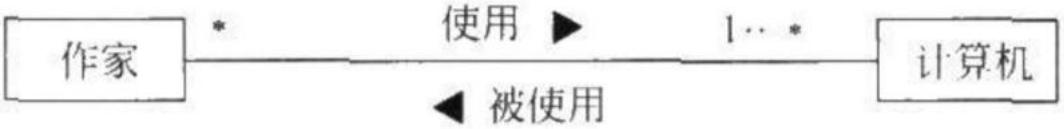
普通关联是最常见的关联关系，只要在类与类之间存在连接关系就可以用普通关联表示

②：表示

第一，普通关联的图示符号是连接两个类之间的直线，如下图

第二，关联是双向的，可为关联起一个名字。在名字前面(或后面)加一个表示关联方向的黑三角

第三，在表示关联的直线两端可以写上重数，它表示该类有多少个对象与对方的一个对象连接。未明确标出关联的重数，则默认重数是 1



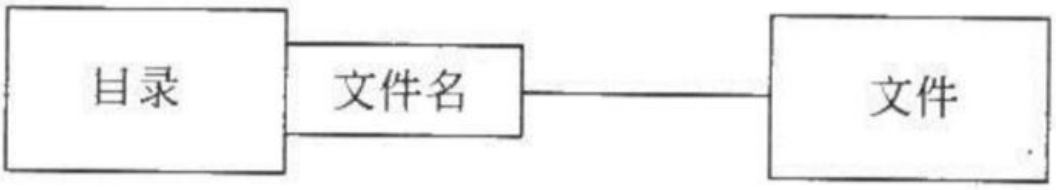
D：限定关联

①：定义

限定关联通常用在一对多或多对多的关联关系中，可以把模型中的重数从一对多变成一对一， 或从多对多简化成多对一

②：表示

在类图中把限定词放在关联关系末端的一个小方框内。



利用限定词“文件名”表示了目录与文件之间的关系，利用限定词把一对多关系简化成了一对一关系

③：意义

限定提高了语义精确性，增强了查询能力

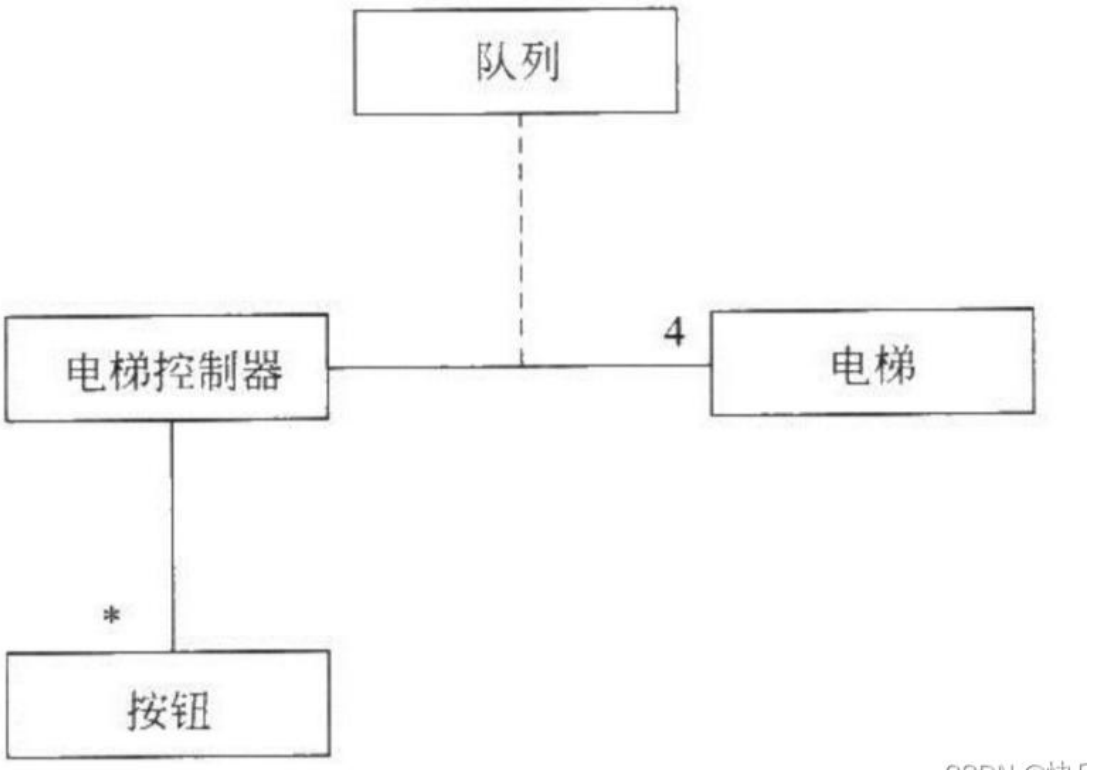
E：关联类

①：定义

为了说明关联的性质，可能需要一些附加信息。关联类可以用来记录相关信息

②：表示

关联类通过一条虚线与关联连接



关联中的每个连接与关联类的一个对象相联系

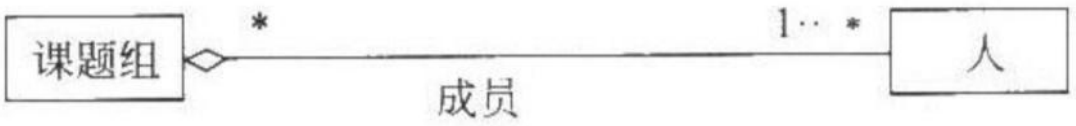
（2）聚集（它是关联的特例）

聚集(聚合)是关联的特例。表示类与类之间的关系是整体与部分的关系。在陈述需求时使用的**“包含”、“组成”、“分为…部分”**等字句，往往意味着存

在聚集关系。除了一般聚集之外，还有两种特殊的聚集关系，分别是共享聚集和组合聚集

A：共享聚集

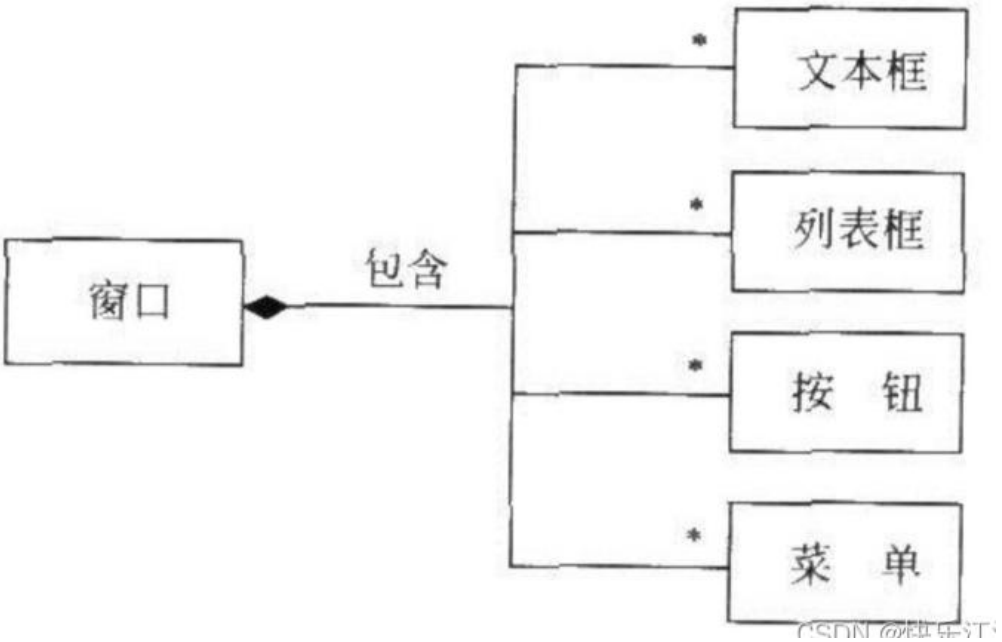
如果在聚集关系中处于部分方的对象可同时参与多个处于整体方对象的构成，则该聚集称为共享聚集



一般聚集和共享聚集的图示符号，都是在表示关联关系的直线末端紧挨着整体类的地方画一个空心菱形

B：组合聚集

如果部分类完全隶属于整体类，部分与整体共存，整体不存在了部分也会随之消失， 则该聚集称为组合聚集(组成)



组成关系用实心菱形示例

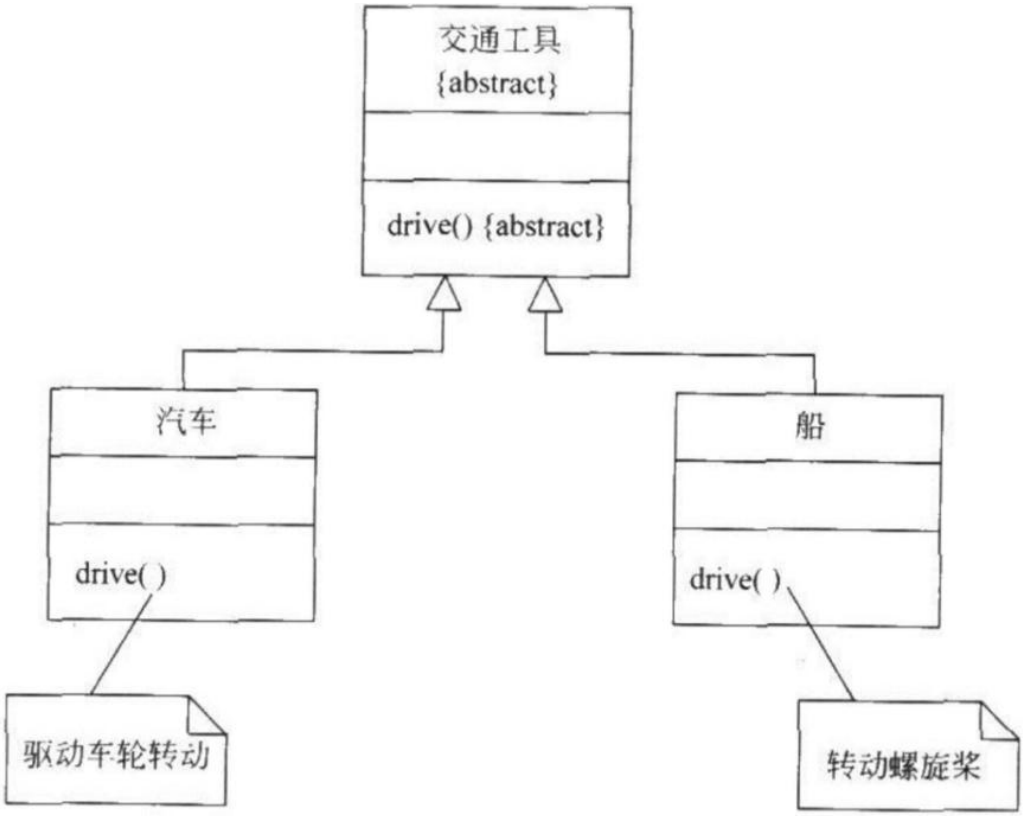
（3）泛化（本质就是继承）

UML 中的泛化关系就是继承关系，它是通用元素和具体元素之间的一种分类关系。具体元素完全拥有通用元素的信息，并且还可以附加一些其他信息。在 UML 中，用一端为空心三角形的连线表示泛化关系，三角形的顶角紧挨着通用元素

A：普通泛化

①：抽象类

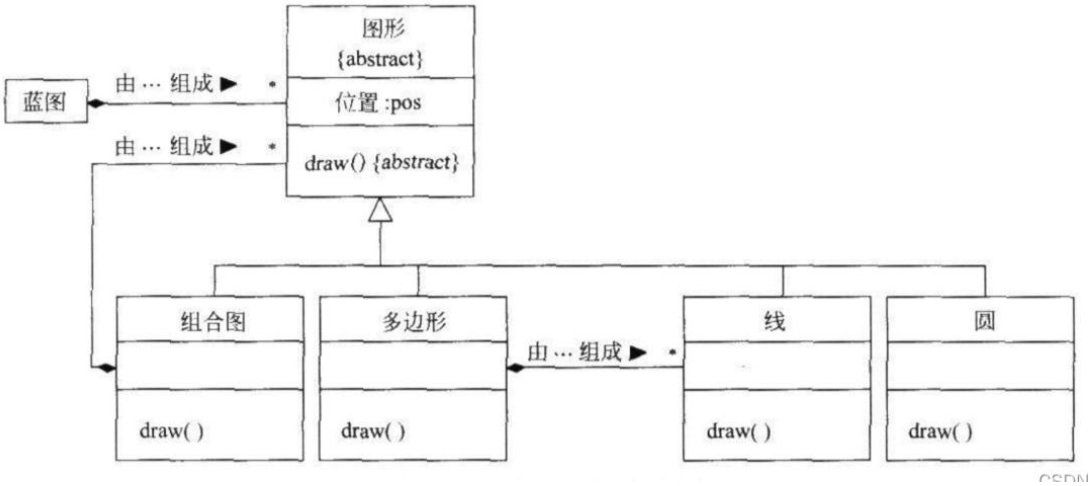
没有具体对象的类称为抽象类。抽象类通常都有抽象操作，来指定该类的所有子类应具有哪些行为



表示抽象类是在类名下方附加一个标记值{abstract}，表示抽象操作是在操作标记后面跟随一个性质串{abstract}

②：具体类

具体类有自己的对象，并且该类的操作都有具体的实现方法



B：受限泛化

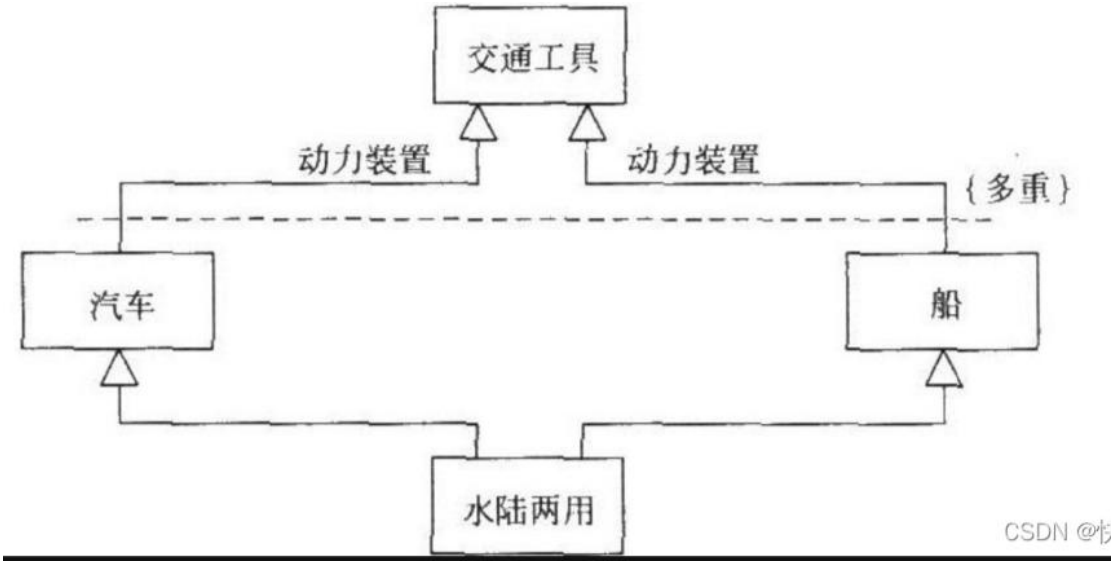
①：定义

可以给泛化关系附加约束条件，以进一步说明该泛化关系的使用方法或扩充方法，这样的泛化关系称为受限泛化

②：约束

预定义的约束有 4 种（都是语义约束）

① 多重：一个子类可以同时多次继承同一个上层基类



② 不相交：一个子类不能多次继承同一个基类。-般的继承都是不相交继承

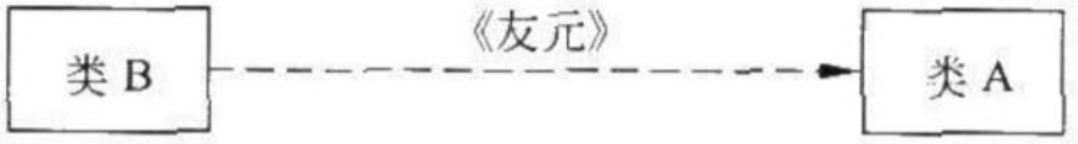
③ 完全：父类的所有子类都已在类图中穷举出来了

④ 不完全：父类的子类并没有都穷举出来，随着对问题理解的深入，可不断补充和维护。是默认的继承关系

（4）依赖和细化

A：依赖关系

依赖关系描述两个模型元素之间的语义连接关系:其中一个模型元素是独立的，另一个模型元素不是独立的，它依赖于独立的模型元素，如果独立的模型元素改变了，将影响依赖于它的模型元素



在 UML 类图中用带箭头的虚线连接有依赖关系的两个类，箭头指向独立的类。在虚线上可以带一个版类标签，具体说明依赖的种类

B：细化关系

对同一个事物在不同抽象层次上描述时，这些描述之间具有细化关系

细化的图示符号为由元素 B 指向元素 A 的一端为空心三角形的虚线

第四节：面向对象建模之动态模型和功能模型

一：动态模型

（1）概念

动态模型表示瞬时的、行为化的系统的控制性质，它规定了对象模型中的对象的合法变化序列

（2）建模

用 UML 提供的状态图来描绘对象的状态、触发状态转换的事件以及对象的行为。每个类的动态行为用一张状态图来描绘，各个类的状态图通过共享事件合并起来，从而构成系统的动态模型，即动态模型是基于事件共享而互相关联的一组状态图的集合

二：功能模型

（1）概念

A：定义

功能模型表示变化的系统的功能性质，它指明了系统应该做什么，因此更直接地反映了用户对目标系统的需求

B：组成

功能模型由一组数据流图组成

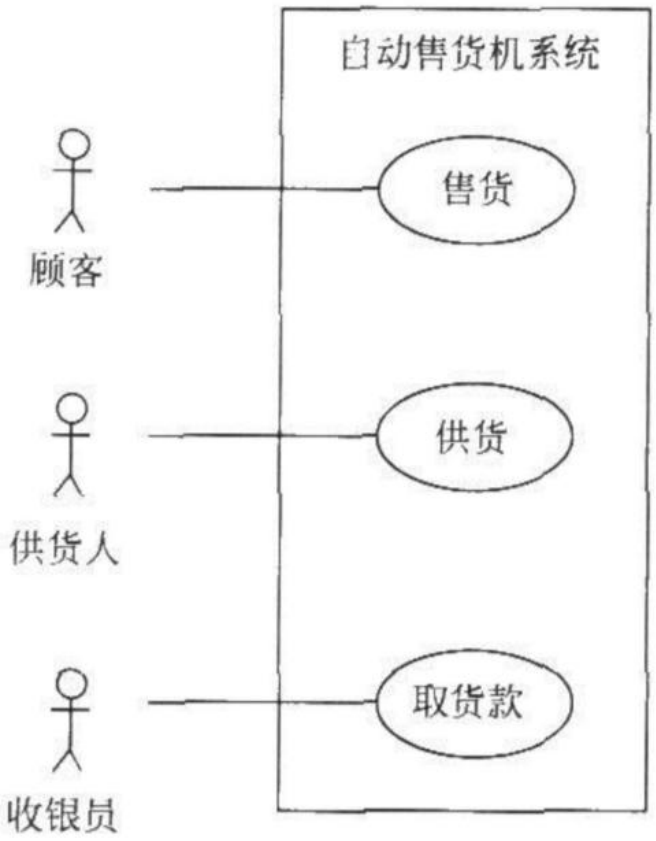
（2）用例图

UML 提供的用例图也是进行需求分析和建立功能模型的强有力工具。在 UML 中把用用例图建立起来的系统模型称为用例模型

A：定义

用例模型描述的是外部行为者所理解的系统功能。用例模型的建立是系统开发者和用户反复讨论的结果，它描述了开发者和用户对需求规格所达成的共识。

B：表示：



自动售货机系统用例图 CSDN

①：系统

定义：系统被看作是一个提供用例的黑盒子，内部如何工作、用例如何实现，这些对于建立用例模型来说都是不重要的

表示：系统用方框表示，其边线表示系统的边界，用于划定系统的功能范围，定义了系统所具有的功能。描述该系统功能的用例置于方框内，代表外部实体的行为者置于方框外

②：用例

定义：一个用例是可以被行为者感受到的、系统的一个完整的功能。在 UML 中把用例定义成系统完成的一系列动作

表示：在 UML 中，椭圆代表用例。用例通过关联与行为者连接，关联指出一个用例与哪些行为者交互，这种交互是双向的

特征：

① 用例代表某些用户可见的功能，实现一个具体的用户目标

② 用例总是被行为者启动的，并向行为者提供可识别的值

③ 用例必须是完整的

注意：用例是一个类，它代表一类功能而不是使用该功能的某个具体实例。用例的实例是系统的一种实际使用方法，通常把用例的实例称为脚本。脚本是系统的一次具体执行过程

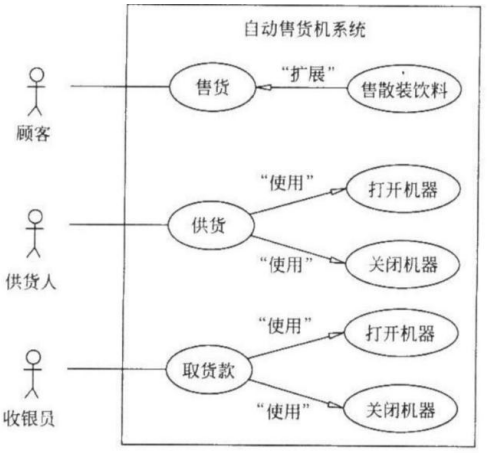
③：行为者

定义：行为者是指与系统交互的人或其他系统，它代表外部实体。使用用例并且与系统交互的任何人或物都是行为者。行为者代表一种角色，而不是某个具体的人或物

表示：在 UML 中，线条人代表行为者。在用例图中用直线连接行为者和用例，表示两者之间交换信息，称为通信联系。行为者触发用例，并与用例交换信息。单个行为者可与多个用例联系，一个用例也可与多个行为者联系

④：用例间关系

扩展关系：向一个用例中添加一些动作后构成了另一个用例，这两个用例之间的关系就是扩展关系，后者继承前者的一些行为，通常把后者称为扩展用例



含扩展和使用关系的用例图

使用关系：一个用例使用另一个用例时，这两个用例之间就构成了使用关系

两种关系的异同

① 都是从几个用例中抽取那些公共的行为并放入一个单独的用例中,而这个用例被其他用例使用或扩展

② 使用和扩展的目的是不同的。在描述一般行为的变化时采用扩展关系

③ 在两个或多个用例中出现重复描述又想避免这种重复时，采用使用关系

三、三种模型比较（了解）

① 针对每个类建立的动态模型，描述了类实例的生命周期或运行周期

② 状态转换驱使行为发生，这些行为在数据流图中被映射成处理，在用例图中被映射成用例，它们同时与类图中的服务相对应

③ 功能模型中的处理对应于对象模型中的类所提供的服务

④ 数据流图中的数据存储，以及数据的源点/终点，通常是对象模型中的对象

⑤ 数据流图中的数据流，往往是对象模型中对象的属性值，也可能是整个对象

⑥ 用例图中的行为者，可能是对象模型中的对象

⑦ 功能模型中的处理可能产生动态模型中的事件

⑧ 对象模型描述了数据流图中的数据流、数据存储以及数据源点/终点的结构

第九章：面向对象分析

第一节：面向对象分析的基本过程和需求陈述

一：面向对象分析的基本过程

（1）定义

面向对象分析：就是抽取和整理用户需求并建立问题域精确模型的过程

（2）3 个子模型与 5 个层次

A：3 个子模型

面向对象建模得到的模型包含系统的三个要素：

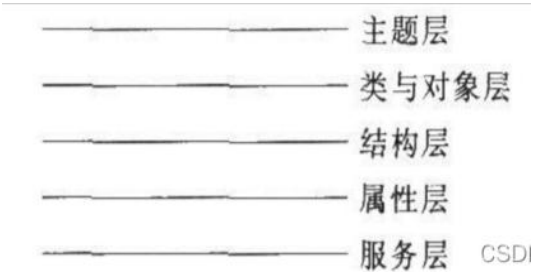
静态结构（对象模型）：解决任一问题，都需要从客观世界实体及实体间相互关系抽象出极有价值的对象模型

交互次序（动态模型）：当问题涉及交互作用和时序时，动态模型是重要的

数据变换（功能模型）：解决运算量很大的问题，则涉及重要的功能模型

B：5 个层次

复杂问题的对象模型由 5 个层次组成:主题层、类与对象层、结构层、属性层和服务层。这 5 个层次一层比一层显现出对象模型的更多细节



①：主题

主题是指导读者理解大型、复杂模型的一种机制。即通过划分主题把一个大型、复杂的对象模型分解成几个不同的概念范畴

②：7 ± 2 原则

心理研究表明，人类的短期记忆能力一般限于一次记忆 5~9 个对象，面向对象从两个方面体现这条原则

控制可见性：控制读者能见到的层次数目来控制可见性

指导读者注意力：增加了主题层，可从高层次描述总体模型，并指导读者的注意力

③：面向对象分析顺序

1) 寻找类与对象
2) 识别结构
3) 识别主题
4) 定义属性
5) 建立动态模型
6) 建立功能模型
7) 定义服务

④：建模要点

1) 面向对象分析不可能严格按照顺序线性进行
2) 必须在反复分析中对初始模型中不准确、不完整和错误的内容加以扩充和更正
3) 仔细研究类似问题的分析结果，尽可能重用这些结果

二：需求陈述

（1）内容

① 阐明“做什么”而不是“怎样做”
② 描述用户的需求而不是提出解决问题的方法
③ 指出哪些是系统必要的性质，哪些是任选的性质
④ 避免对设计策略施加过多的约束，不描述系统的内部结构
⑤ 描述系统性能及系统与外界环境交互协议
⑥ 描述采用的软件工程标准、模块构造准则、将来的扩充以及可维护性要求等方面

（2）书写规范

① 做到语法正确，而且应该慎重选用名词、动词、形容词和同义词
② 必须把需求与实现策略区分开，后者不是问题域的本质性质
③ 需求陈述可简可繁
④ 避免出现具有二义性的、不完整的、不一致的内容

第二节：面向对象分析之建立对象模型

一： 基本概念

（1）对象模型

面向对象分析的首要工作就是建立问题域的对象模型。对象模型表示静态的、结构化的系统的数据性质。它是对模拟客观世界实体的对象以及对象彼此间的关系的映射，描述了系统的静态结构。对象模型为建立动态模型和功能模型，提供了实质性的框架

（2）先建立对象模型的原因

① 静态数据结构对应用细节依赖较少，比较容易确定

② 当用户的需求变化时，静态数据结构相对来说比较稳定

（3）信息来源

需求陈述、应用领域的专业知识、客观世界的常识，是建立对象模型时的主要信息来源

（4）典型的建模步骤

① 确定对象类和关联(对于大型复杂问题还要进一步划分出若干个主题);

② 给类和关联增添属性，以进一步描述它们;

③ 使用适当的继承关系进一步合并和组织类

二：确定类与对象

（1）找出候选的类与对象

A：客观事物分类

对象是对问题域中有意义的事物的抽象，它们既可能是物理实体，也可能是抽象概念。客观事物可分为下述 5 类:

① 可感知的物理实体;

② 人或组织的角色;

③ 应该记忆的事件;

④ 两个或多个对象的相互作用;

⑤ 需要说明的概念

B：非正式分析

以用自然语言书写的需求陈述为依据，把陈述中的名词作为类与对象的候选者,用形容词作为确定属性的线索，把动词作为服务的候选者。这种方法确定的候选者是非常不准确的，其中往往包含大量不正确或不必要的事物，需要经过更进一步的严格筛选

C：提取隐含的类与对象

（2）筛选出正确的类与对象

如果两个类表达了同样的信息，则应该保留在此问题域中最富于描述力的名称

需要把与本问题密切相关的类与对象放进目标系统中

系统无须记忆笼统的、泛指的名词信息，把这些笼统的或模糊的类去掉

把描述的是其他对象属性的词从候选类与对象中去掉

慎重考虑既可作为名词，又可作为动词的词，以便正确地决定把它们作为类还是作为类中定义的操作。本身具有属性，需独立存在的操作，应该作为类与对象

应该去掉仅和实现有关的候选的类与对象

三：确定关联

（1）关联

A：定义

两个或多个对象之间的相互依赖、相互作用的关系就是关联。在需求陈述中使用的描述性动词或动词词组,通常表示关联关系

B：确定关联的重要性

分析确定关联，能促使分析员考虑问题域的边缘情况，有助于发现尚未被发现的类与对

象

（2）步骤

A：初步确定关联

① 直接提取动词短语得出的关联

② 需求陈述中隐含的关联

③ 根据问题域知识得出的关联

B：筛选

筛选时主要根据下述标准删除候选的关联:

已删去的类之间的关联：如果在分析确定类与对象的过程中已经删掉了某个候选类，则与这个类有关的关联也应该删去，或用其他类重新表达这个关联

与问题无关的或应在实现阶段考虑的关联：应该把处在本问题域之外的关联与实现密切相关的关联删去

瞬时事件：关联应该描述问题域的静态结构，而不应该是一个瞬时事件

三元关联：三个或三个以上对象间的关联，可分解为二元关联或用词组描述成限定的关联

派生关联：去掉那些可以用其他关联定义的冗余关联

C：改进

可以从以下几个方面进一步完善经筛选后余下的关联

① 正名：仔细选择含义更明确的名字作为关联名

② 分解：为了能够适用于不同的关联，必要时应该分解以前确定的类与对象

③ 补充：发现了遗漏的关联就应该及时补上

④ 标明重数：应该初步判定各个关联的类型，并粗略地确定关联的重数

四：划分主题

（1）定义

在开发大型、复杂系统的过程中，为了降低复杂程度,把系统再进一步划分成几个不同的主题，即在概念上把系统包含的内容分解成若干个范畴

（2）针对不同类型的方法

① 规模小的系统：可能无须引入主题层

② 含有较多对象的系统：首先识别出类与对象和关联，然后划分主题，并用它作为指导开发者和用户观察整个模型的一种机制

③ 规模大的系统：首先由高级分析员粗略地识别对象和关联，然后初步划分主题，经进一步分析，对系统结构有更深入的了解之后，再进一步修改和精炼主题

（3）原则

① 按问题领域而不是用功能分解方法来确定主题

② 按照使不同主题内的对象相互间依赖和交互最少的原则来确定主题

五：确定属性

（1）属性

属性是对象的性质，借助于属性人们能对类与对象和结构有更深入更具体的认识

注意:在分析阶段不要用属性来表示对象间的关系，使用关联能够表示两个对象间的任何关系，而且把关系表示得更清晰、更醒目

（2）确定属性的步骤

A：分析

在需求陈述中用名词词组表示属性，用形容词表示可枚举的具体属性

借助于领域知识和常识分析需要的属性

仅考虑与具体应用直接相关的属性，不要考虑那些超出所要解决的问题范围的属性

首先找出最重要的属性，以后再逐渐把其余属性增添进去

不要考虑那些纯粹用于实现的属性

B：选择

从初步分析确定下来的属性中删掉不正确的或不必要的属性。有以下几种常见情况:

误把对象当作属性：如果某个实体的独立存在比它的值更重要，则应把它作为一个对象而不是对象的属性

误把关联类的属性当作一般对象的属性：如果某个性质依赖于某个关联链的存在，则该性质是关联类的属性，在分析阶段不应把它作为一般对象的属性

把限定误当成属性：如果把某个属性值固定下来以后能减少关联的重数，则应该考虑把这个属性重新表达成一个限定词。

误把内部状态当成了属性：如果某个性质是对象的非公开的内部状态，则应该从对象模型中删除这个属性。

过于细化：在分析阶段应该忽略那些对大多数操作都没有影响的属性

存在不一致的属性：类应该是简单而且一致的。如果得出一些看起来与其他属性毫不相关的属性，则应该考虑把类分解成两个不同的类

六：识别继承关系

（1）建立继承关系的方式

确定了类中应该定义的属性之后，就可以利用继承机制共享公共性质，并对系统中众多的类加以组织。可以使用以下两种方式建立继承关系

自底向上：抽象出现有类的共同性质泛化出父类，这个过程实质上模拟了人类归纳思维的过程

自顶向下：把现有类细化成更具体的子类,这模拟了人类的演绎思维过程。从应用域中常常能明显看出应该做的自顶向下的具体化工作

（2）多重继承

A：作用

利用多重继承可以提高共享程度，但增加了概念上以及实现时的复杂程度

B：要点

指定一个主要父类，从它继承大部分属性和行为;

次要父类只补充一些属性和行为

七：反复修改

（1）必要性

软件开发过程就是一个多次反复修改、逐步完善的过程。仅仅经过一次建模过程很难得到完全正确的对象模型

（2）面向对象在修改时的优点

面向对象的概念和符号在整个开发过程中都是一致的，比使用结构分析、设计技术更容易实现反复修改、逐步完善的过程

第三节：面向对象分析之建立动态模型和功能模型

一：建立动态模型

（1）概念

A：适用性

① 对于仅存储静态数据的系统来说，动态模型并没有什么意义

② 在开发交互式系统时，动态模型却起着很重要的作用

③ 收集输入信息是系统的主要工作时，则在开发时建立正确的动态模型是至关重要的

B：步骤

① 编写典型交互行为的脚本

② 从脚本中提取出事件，确定触发每个事件的动作对象以及接受事件的目标对象

③ 排列事件发生的次序，确定每个对象的状态及状态间的转换关系，用状态图描绘

④ 比较各个对象的状态图，确保事件之间的匹配

（2）编写脚本

A：定义

脚本是指系统在某一执行期间内出现的一系列事件。脚本描述用户与目标系统之间的一个或多个典型的交互过程。编写脚本的过程，就是分析用户对系统交互行为的要求的过程

B：目的

保证不遗漏重要的交互步骤，有助于确保交互过程的正确性、清晰性

C：内容

脚本描写的范围主要由编写脚本的具体目的决定，既可以包括系统中发生的全部事件，也可以只包括由某些特定对象触发的事件

D：方法

① 编写正常情况的脚本

② 考虑特殊情况

③ 考虑出错情况

（3）设想用户界面

大多数交互行为都可以分为应用逻辑和用户界面两部分，通常，系统分析员首先集中精力考虑系统的信息流和控制流，而不是首先考虑用户界面

A：重要性

用户界面的美观程度、方便程度、易学程度以及效率等，是用户使用系统时最先感受到的。用户界面的好坏往往对用户是否喜欢、是否接受一个系统起很重要的作用

B：目的

这个阶段用户界面的细节并不太重要，重要的是在这种界面下的信息交换方式。目的是确保能够完成全部必要的信息交换，而不会丢失重要的信息

C：方法

快速地建立起用户界面的原型，供用户试用与评价

（4）画事件跟踪图

A：必要性

用自然语言书写的脚本往往不够简明，而且有时在阅读时会有二义性。为了有助于建立动态模型，需要画出事件跟踪图

B：步骤

①：确定事件

1.提取出所有外部事件

找出正常事件、异常事件和出错条件(传递信息的对象的动作也是事件)

把对控制流产生相同效果的事件组合为一类事件，并取一个唯一的名字

2.画出事件跟踪图

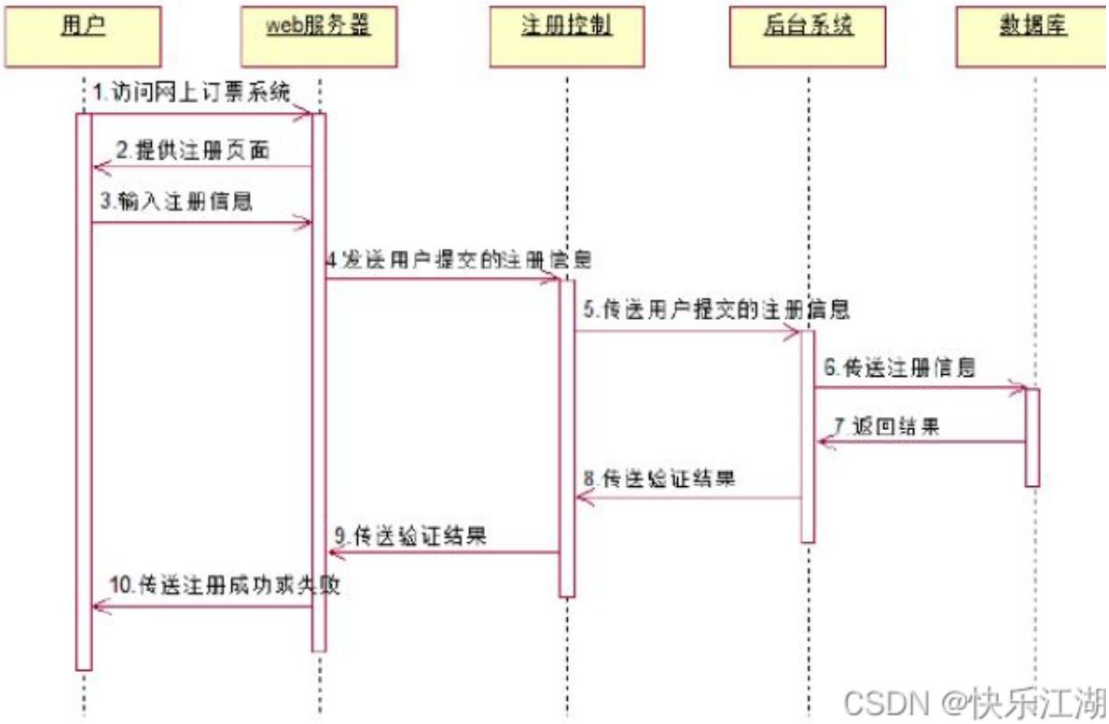
一条竖线代表一个对象

每个事件用一条水平的箭头线表示

箭头方向从事件的发送对象指向接受对象

时间从上向下递增

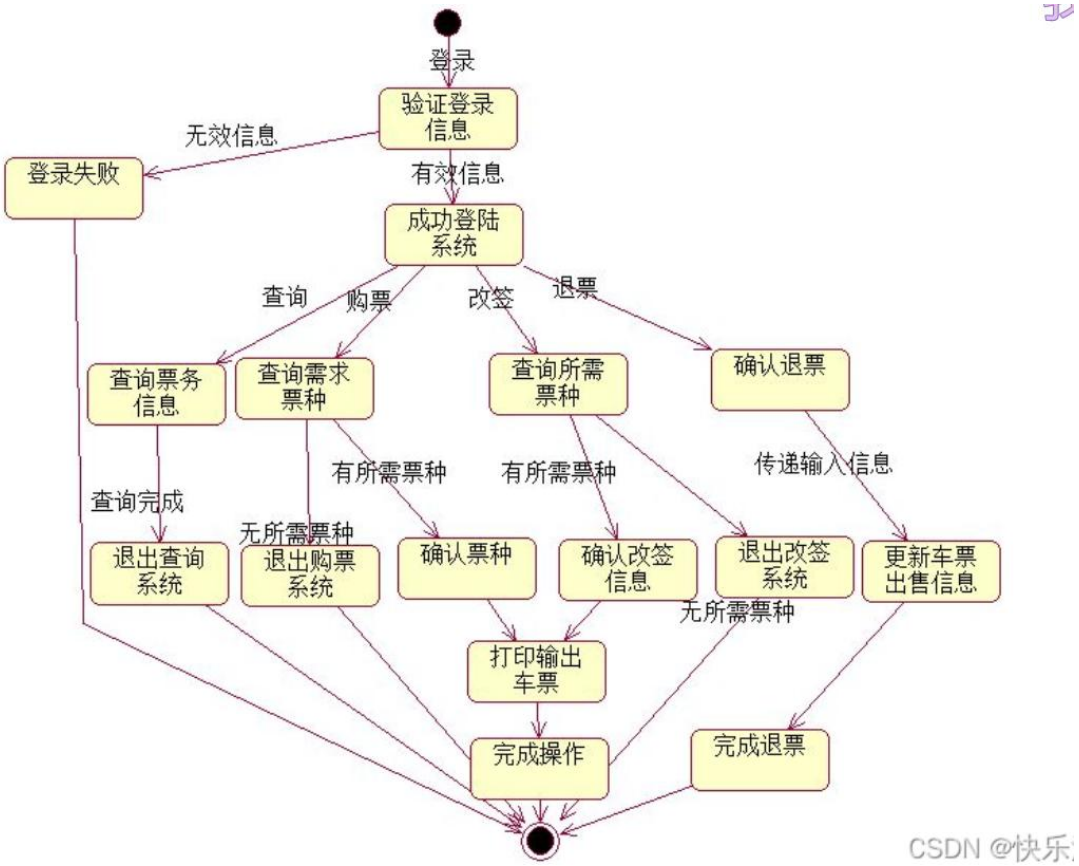
用箭头线在垂直方向上的相对位置表示事件发生的先后，不表示事件间的时间差



（5）画状态图

A：定义

状态图描绘事件与对象状态的关系。当对象接受了一个事件以后，它的下个状态取决于当前状态及所接受的事件。由事件引起的改变称为“转换”。一张状态图描绘了一类对象的行为，它确定了由事件序列引出的状态序列



B：适用性

对于仅响应与过去历史无关的那些输入事件,或者把历史作为不影响控制流的参数类的对象，状态图是不必要的

C：方法

① 仅考虑事件跟踪图中指向某条竖线的那些箭头线。把这些事件作为状态图中的有向边，边上标以事件名

② 两个事件之间的间隔就是一个状态，每个状态取个有意义的名字。从事件跟踪图中当前考虑的竖线射出的箭头线，是这条竖线代表的对象达到某个状态时所做的行为。

③ 根据一张事件跟踪图画出状态图后，再把其他脚本的事件跟踪图合并到该图中

④ 考虑完正常事件后再考虑边界情况和特殊情况，包括在不适当时候发生的事件

（6）审查动态模型

检查系统级的完整性和一致性

审查每个事件，跟踪它对系统中各对象所产生的效果，保证与每个脚本都匹配

二：建立功能模型

（1）定义

功能模型表明了系统中数据之间的依赖关系，以及有关的数据处理功能，它由一组数据流图组成。在建立了对象模型和动态模型之后再建立功能模型

（2）画出基本系统模型图

基本的系统模型有下述两部分组成:

数据源点/终点：数据源点输入的数据和输出到数据终点的数据，是系统

与外部世界间交互事件的参数。

处理框：代表了系统加工、变换数据的整体功能。

（3）画出功能级数据流图

把基本系统模型中单一的处理框分解成若干个处理框，以描述系统加工、变换数据的基本功能，就得到功能级数据流图

（4）描述处理框功能

A：要点

着重描述每个处理框所代表的功能，而不是实现功能的具体算法

B：分类

① 说明性描述（更为重要）：规定了输入值和输出值之间的关系，以及输出值应遵循的规律

② 过程性描述：通过算法说明“做什么”

第十章：面向对象设计

第一节：面向对象设计的基本概念与准则

一：面向对象设计概念

（1）定义

设计是把分析阶段得到的需求转变成符合成本和质量要求的、抽象的系统实现方案的过程。从面向对象分析到面向对象设计是一个逐渐扩充模型的过程，即面向对象设计就是用面向对象观点建立求解域模型的过程

（2）设计与分析的关系

① 分析结果可以直接映射成设计结果，而在设计过程中又会加深和补充对系统需求的理解，进一步完善分析结果

② 分析和设计活动是一个多次反复迭代的过程

③ 分析是提取和整理用户需求，并建立问题域精确模型的过程。设计则是把分析阶段得到的需求转变成符合成本和质量要求的、抽象的系统实现方案的过程

（3）分类

① 系统设计:确定实现系统的策略和目标系统的高层结构

② 对象设计:确定解空间中的类、关联、接口形式及实现服务的算法

二：面向对象的设计准则

（1）模块化

面向对象软件开发模式支持了把系统分解成模块设计的原理,对象是面向对象软件系统中的模块，它是把数据结构和操作这些数据的方法紧密地结合在一起所构成的模块

（2）抽象

① 面向对象的程序设计语言不仅支持过程抽象，而且支持数据抽象，对象类实际上是具有继承机制的抽象数据类型，它对外开放的公共接口构成了类的规格说明(协议)，这种接口规定了外界可以使用的合法操作符，利用这些操作符可以对类实例中包含的数据进行操作

② 规格说明抽象：使用者无须知道操作符的实现算法和类中数据元素的具体表示方法，就可以通过这些操作符使用类中定义的数据，这种抽象称为规格说明抽象

③ 参数化抽象：指当描述类的规格说明时并不具体指定所要操作的数据类型，

而是把数据类型作为参数，使得类的抽象程度更高，应用范围更广，可重用性更高

（3）信息隐藏

在面向对象的软件中，信息隐藏通过对象的封装来实现，即类结构分离了接口与实现，从而支持了信息隐藏。对于类，属性的表示方法和操作的实现算法都是隐藏的

（4）低耦合

A：交互耦合

对象间的耦合通过消息连接来实现，则这种耦合是交互耦合。要使交互耦合尽可能松散,必须遵守下述准则。

尽量降低消息连接的复杂程度。

尽量减少消息中包含的参数个数，降低参数的复杂程度

减少对象发送或接收的消息数

B：继承耦合

继承是一般类与特殊类之间耦合的一种形式。通过继承关系结合起来的基类和派生类构成了系统中粒度更大的模块，因此，它们彼此之间应该结合得越紧密越好

（5）高内聚

A：服务内聚

一个服务应该完成一个且仅完成一个功能

B：类内聚

设计类的准则是，一个类应该只有一个用途，它的属性和服务应该是高内聚的。如果某个类有多个用途，应该把它分解成多个专用的类

C：一般一特殊内聚

设计出的一般一特殊结构应该是对相应的领域知识的正确抽取。紧密的继承耦合与高度的一般一特殊内聚是一致的

（6）可重用

尽量使用已有的类

如果需要创建新类，则在设计这些新类的协议时应该考虑将来的可重复使用性

第二节：启发规则和软件重用

一：启发规则

（1）设计结果应该清晰易懂

保证设计结果应该清晰易懂的主要因素如下

① 用词一致：应该使名字与它所代表的事物－致,而且应该尽量使用人们习惯的名字。不同类中相似服务的名字应该相同

② 使用已有的协议：如果开发同一软件的其他设计人员已经建立了类的协议，或者在所使用的类库中已有相应的协议，则应该使用这些已有的协议

③ 减少消息模式的数目：如果已有标准的消息协议，设计人员应该遵守这些协议

④ 避免模糊的定义：一个类的用途应该是有限的，而且应该从类名可以较容易地推出它的用途

（2）一般一特殊结构的深度适当

① 使类等级中包含的层次数适当，类等级中包含的层次保持在 7±2

② 不能仅从方便编码的角度出发随意创建派生类，应该使一般一特殊结

构与领域知识或常识保持一致

（3）设计简单的类

避免包含过多属性：属性过多通常表明这个类过分复杂了，它所完成的功能可能太多了

有明确的定义：为了使类的定义明确，分配给每个类的任务应该简单，最好能使用一两个简单语句描述它的任务

简化对象之间的合作关系：如果需要多个对象协同配合才能做好一件事，则破坏了类的简明性和清晰性

不要提供太多服务：一个类提供的服务过多，同样表明这个类过分复杂。典型地，一个类提供的公共服务不超过 7 个

划分“主题”

（4）使用简单的协议

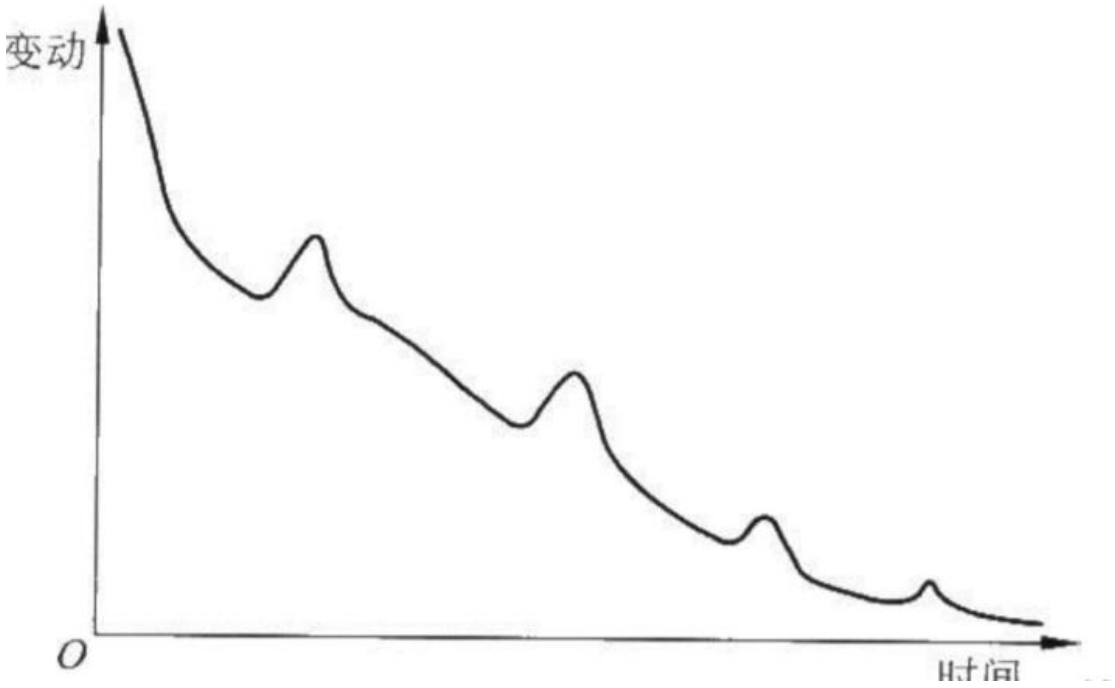
（5）使用简单的服务

避免使用复杂的服务

需要在服务中使用 CASE 语句时，应用一般一特殊结构代替这个类

（6）把设计变动减至最小

理想的设计变动曲线如下图所示。即在设计的早期阶段,变动较大，随着时间推移，设计方案日趋成熟改动也越来越小了



二：软件重用

（1）概述

A：重用

重用也叫再用或复用，是指同一事物不作修改或稍加改动就多次重复使用。

软件重用可分为以下 3 个层次:

① 知识重用;

② 方法和标准的重用;

③ 软件成分的重用

B：软件成分重用的级别

①：代码重用

源代码剪贴：是最原始的重用形式，复制或修改源代码时可能出错，且存在严重的配置管理问题

源代码包含：配置管理问题有所缓解，所有包含它的程序都必须重新编译

继承：无须修改已有的代码，就可扩充或具体化在库中找出的类。基本上不存在配置管理问题

②：设计结果重用

重用某个软件系统的设计模型(求解域模型)，有助于把一个应用系统移植到完全不同的软硬件平台上

③：分析结果重用

是一种更高级别的重用，重用某个系统的分析模型。特别适用于用户需求未改变，但系统体系结构发生了根本变化的场合

C：典型的可重用软件成分

① 项目计划：软件项目计划的基本结构和许多内容都是可以跨项目重用的

② 成本估计：在只做极少修改或根本不做修改的情况下，重用对该功能的成本估计结果

③ 体系结构：创建一组类属的体系结构模板，并把那些模板作为可重用的设计框架

④ 需求模型和规格说明：类和对象的模型、规格说明、用传统软件工程方法开发的分析模型是重用的候选者

⑤ 设计：用传统方法开发的体系结构、数据、接口和过程设计结果，是重用的候选者

⑥ 源代码：用兼容的程序设计语言书写的、经过验证的程序构件，是重用的候选者

⑦ 用户文档和技术文档：即使针对的应用是不同的，也经常有可能重用用户文档和技术文档的大部分

⑧ 用户界面：这可能是最广泛被重用的软件成分，GUI (图形用户界面)软件经常被重用。因为它可占到一个应用程序的 60%的代码量

⑨ 数据：被重用的数据包括:内部表、列表和记录结构，以及文件和完整的数据库

⑩ 测试用例：一旦设计或代码构件将被重用，相关的测试用例应该“附属于”它们也被重用

（2）类构件

A：可重用软件构件特点

① 模块独立性强：具有单一、完整的功能，且经过反复测试被确认是正确的。它应该是一个不受或很少受外界干扰的封装体，其内部实现在外面是不可见的

② 具有高度可塑性：可重用的软构件必须具有高度可裁剪性，即必须提供为适应特定需求而扩充或修改已有构件的机制，而且所提供的机制必须使用起来简单方便

③ 接口清晰、简明、可靠：软件构件应该提供清晰、简明、可靠的对外接口，而且还应该有详尽的文档说明，以方便用户使用

B：类构件的重用方式

类构件：面向对象技术中的“类”，是比较理想的可重用软构件

①：实例重用

除了用已有的类为样板直接创建该类的实例之外,还可以用几个简单的对象作为类的成员创建出一个更复的类

②：继承重用

当已有的类构件不能通过实例重用方式满足当前系统的需求时,利用继承机制从已有类派生出符合需要的子类，是安全修改已有的类构件并获得可在当前系统中使用的类构件的有效手段

③：多态重用

在设计类构件时应把注意力集中在下列这些可能妨碍重用的操作上：

① 与表示方法有关的操作

② 与数据结构、数据大小等因素有关的操作

③ 与外部设备有关的操作

④ 实现算法在将来可能会改变的核心操作

（3）软件重用的效益

A：质量

随着每一次重用， 都会有一些错误被发现并被清除，构件的质量也会随之改善。随着时间的推移，构件将变成实质上无错误的。重用给软件产品的质量和可靠性带来实质性的提高

B：生产率

把可重用的软件成分应用于软件开发的全过程时，创建计划、模型、文档、代码和数据所花费的时间将减少，从而用较少的投入给客户提供相同级别的产品，故生产率得到了提高

C：成本

软件重用带来的净成本节省可以用下式估算：

$$C = C_s - C_r - C_d$$

- C_s：是**项目从头开发时**所需要的成本
- C_r：是与重用相关联的成本
- C_d：是交付给客户的软件的实际成本

与重用相关联的成本 C，主要包括下述成本:

①领域分析与建模的成本

②设计领域体系结构的成本

③为方便重用而增加的文档的成本

④维护和完善可重用的软件成分的成本

⑤为从外部获取构件所付出的版税和许可证费用

⑥创建及运行重用库的费用

⑦对设计和实现可重用构件的人员的培训费用

第三节：系统分解和设计问题域子系统

一：系统分解

（1）分解思想

在设计比较复杂的应用系统时，先把系统分解成若干个较小部分，然后分别设计每个部分。这样做有利于降低设计的难度，有利于分工协作，也有利于维护人员对系统理解和维护

（2）子系统

A：定义

系统的主要组成部分称为子系统，通常根据所提供的功能来划分子系统

B：划分原则

根据所提供的功能来划分子系统，子系统数目应该与系统规模基本匹配

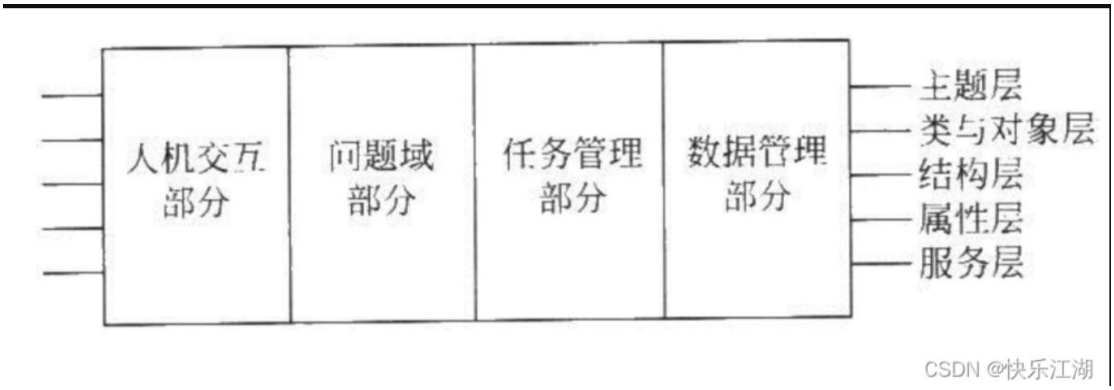
各个子系统之间应该具有尽可能简单、明确的接口

应该尽量减少子系统彼此间的依赖性

（3）分解面向对象设计模型

A：表示

典型的面向对象设计模型如下图所示



CSDN @快乐江湖

面向对象设计模型由主题、类与对象、结构、属性、服务 5 个层次组成。这 5 个层次一层比一层表示的细节更多，可以把这 5 个层次想象为整个模型的水平切片

面向对象设计模型在逻辑上都由 4 大部分组成，分别对应于组成目标系统的 4 个子系统,即问题域子系统、人机交互子系统、任务管理子系统和数据管理子系统

B：子系统间交互方式

①：客户-供应商关系（Client-supplier）

作为“客户”的子系统调用作为“供应商”的子系统，后者完成某些服务工作并返回结果。作为客户的子系统必须了解作为供应商的子系统的接口，后者却无须了解前者的接口

②：平等伙伴关系

每个子系统都可能调用其他子系统，每个子系统都必须了解其他子系统的接口。由于各个子系统需要相互了解对方的接口，子系统之间的交互复杂，且还可能存在通信环路

C：组织系统的方案

①：层次组织

软件系统组织成一个层次系统，每层是一个子系统。上层在下层的基础上建立，下层为实现上层功能而提供必要的服务。每一层内所包含的对象，彼此间相互独立，而处于不同层次上的对象，彼此间有关联。在上、下层之间存在客户-供应商关系。低层子系统提供服务，上层子系统使用下层提供的服务

封闭式：每层子系统仅仅使用其直接下层提供的服务。降低了各层次之间的相互依赖性，更容易理解和修改

开放式：子系统可以使用处于其下面的任何一层子系统所提供的服务。优点是减少了需要在每层重新定义的服务数目，使系统更高效更紧凑。但其不符合信息隐藏原则

②：块状组织

把软件系统垂直地分解成若干个相对独立的、弱耦合的子系统，一个子系统相当于一块,每块提供一种类型的服务

③：层次和块状的结合

当混合使用层次结构和块状结构时，同一层次可以由若干块组成，而同一块也可以分为若干层

④：设计系统的拓扑结构

典型的拓扑结构有管道形、树形、星形等。应采用与问题结构相适应的、尽可能简单的拓扑结构，以减少子系统之间的交互数量

二：设计问题域子系统

（1）概念

面向对象分析所得出的问题域精确模型，为设计问题域子系统建立了完整的框架

保持面向对象分析所建立的问题域结构

面向对象设计仅需从实现角度对问题域模型做一-些补充或修改

问题域子系统过分复杂庞大时，应该把它进一步分解成若干个更小的子系统

（2）对问题域模型进行的处理

A：调整需求

①：需要进行调整的情况

用户需求或外部环境发生了变化

分析员对问题域理解不透彻或不能完整、准确地反映用户的真实需求

②：方法

简单地修改面向对象分析结果，然后再把这些修改反映到问题域子系统中

B：重用已有的类

步骤为：

① 在已有类中找出与问题域内某个最相似的类作为被重用的类

② 从被重用的类派生出问题域类

③ 简化对问题域类的定义(从被重用的类继承的属性和服务无须再定义)

④ 修改与问题域类相关的关联，必要时改为与被重用的类相关的关联

C：把问题域类组合在一起

增添一个根类而把若干个问题域类组合在一起

引入根类或基类的办法，可以为一些具体类建立一个公共的协议

D：增添一般化类以建立协议

在设计过程中常常发现，一些具体类需要有一个公共的协议，可以引入附加类以便建立这个协议

E：调整继承层次

如果面向对象分析模型中包含了多重继承关系，然而所使用的程序设计语言却并不提供多重继承机制，则必须修改面向对象分析的结果。具体如下:

多重继承机制：避免出现服务及属性的命名冲突

单继承机制：使用多重继承机制时，必须把面向对象分析模型中的多重继承结构转换成单继承结构

第四节：设计人机交互子系统和设计任务管理子系统

一：设计人机交互子系统

（1）概念

A：主要内容

在面向对象设计过程中,对系统的人机交互子系统进行详细设计，以确定人机交互的细节，其中包括指定窗口和报表的形式、设计命令层次等项内容。

B：重要性

人机界面设计得好，则会使系统对用户产生吸引力，用户在使用系统的过程中会感到兴奋，能够激发用户的创造力，提高工作效率;

人机界面设计得不好，用户在使用过程中就会感到不方便、不习惯，甚至会产生厌烦和恼怒的情绪

（2）设计策略

A：分类用户

应该把将来可能与系统交互的用户按技能水平，或按职务，或按所属集团进行分类

B：描述用户

了解将来使用系统的每类用户的情况，把用户类型、使用目的、特征、关键的成功因素、技能水平、完成本职工作的脚本的信息记录下来

C：设计命令层次

①：研究现有的人机交互含义和准则

设计图形用户界面时，应该遵守广大用户习惯的约定，这样才会被用户接受和喜爱

②：确定初始的命令层次

命令层次实质上是用抽象机制组织起来的、可供选用的服务的表示形式，设计命令层次时，通常先从对服务的过程抽象着手，然后进一步修改它们，以适合具体应用环境的需要

③：精化命令的层次

为进一步修改完善初始的命令层次，应该考虑次序、整体部分关系、宽度和深度等因素

D：设计人机交互类

人机交互类与所使用的操作系统及编程语言密切相关

二：设计任务管理子系统

（1）必要性

许多对象之间往往存在相互依赖关系

在实际使用的硬件中，可能仅由一个处理器支持多个对象

（2）设计步骤

A：分析并发性

①：并发性

如果两个对象彼此间不存在交互，或它们同时接受事件，则它们在本质，上是并发的

②：方法

通过面向对象分析建立起来的动态模型，是分析并发性的主要依据

通过检查各个对象的状态图及它们之间交换的事件，能够把若干个非并发的对象归并到一条控制线中

③：控制线

控制线是一条遍及状态图集合的路径，在这条路径上每次只有一个对象是活动的。在计算机系统中用进程实现控制线。把多个任务的并发执行称为多任务

B：设计任务管理子系统

①：确定事件驱动型任务

某些任务是由事件驱动的，这类任务可能主要完成通信工作,具体任务有

任务处于睡眠状态，等待来自数据线或其他数据源的中断

一旦接收到中断就唤醒该任务，接收数据并把数据放入内存缓冲区或其他目的地，通知需要知道这件事的对象，然后该任务又回到睡眠状态

②：确定时钟驱动型任务

某些任务每隔一定时间间隔就被触发以执行某些处理，具体任务有

任务设置了唤醒时间后进入睡眠状态，等待来自系统的中断

接收到这种中断，任务就被唤醒并做它的工作，通知有关的对象，然后该任务又回到睡眠状态

③：确定优先任务

高优先级：有些服务是优先级的，为了在严格限定的时间内完成，把这类服务分离成独立的任务

低优先级：与高优先级相反，有些服务是低优先级的，属于低优先级处理。设计时用额外的任务把其分离出来

④：确定关键任务

关键任务是有关系统成功或失败的关键处理，这类处理通常都有严格的可靠性要求。处理方法为：在设计过程中用额外的任务把这样的关键处理分离出来，以满足高可靠性处理的要求

⑤：确定协调任务

当系统中存在三个以上任务时，就应该增加一个任务，用它作为协调任务。使用状态转换矩阵可以比较方便地描述该任务的行为。这类任务仅做协调工作,不要让它再承担其他服务工作

⑥：尽量减少任务数

⑦：确定系统资源需求

通过计算系统载荷，来估算所需要的固件的处理能力

综合权衡一致性、成本、性能以及未来的可扩充性和可修改性，决定资源需求

综合考虑各种因素，以决定哪些子系统用硬件实现，哪些子系统用软件实现

第五节：设计数据管理子系统和设计类中的服务

一：设计数据管理子系统

（1）概念

数据库管理子系统：是系统存储或检索对象的基本设施，它建立在某种数据存储管理系统之上，并且隔离了数据存储管理模式的影响

（2）选择数据存储管理模式

A：文件管理系统

优点：文件管理系统是操作系统的一个组成部分，使用它长期保存数据具有成本低和简单的优点

缺点：文件操作的级别低，为提供适当的抽象级别还必须编写额外的代码，不同操作系统的文件管理系统往往有明显差异

B：关系数据库管理系统

优点：

① 理论基础坚实

② 提供了各种最基本的数据管理功能，例如中断恢复，多用户共享，多应用共享，完整性，事务支持等

③ 为多种应用提供了一致的接口

④ 标准化的语言

缺点：

① 运行开销大：即使完成简单的事务，也需要较长的时间

② 不能满足高级应用的需求：关系数据库管理系统是为商务应用服务的，商务应用中数据量虽大但数据结构却比较简单

③ 与程序设计语言的连接不自然: SQL 语言支持面向集合的操作，是一种非过程化的语言;然而大多数程序设计语言本质上却是过程性的，每次只能处理一个记录

C：面向对象数据库管理系统

扩展的关系数据库管理系统：在关系数据库的基础上，增加了抽象数据类型和继承机制，此外还增加了创建及管理类和对象的通用服务

扩展的面向对象程序设计语言：扩充了面向对象程序设计语言的语法和功能，增加了在数据库中存储和管理对象的机制。可以使用统一的面向对象观点进行设计，不需要区分存储数据结构和程序数据结构

（3）设计数据管理子系统

A：设计数据格式

① 文件系统：

定义第一范式表:列出每个类的属性表;把属性表规范成第一范式，从而得到第一范式表的定义

为每个第一范式表定义一个文件

测量性能和需要的存储容量

修改原设计的第一范式，以满足性能和存储需求

② 关系数据库管理系统：

定义第三范式表：列出每个类的属性表;把属性表规范成第三范式，从而得出第三范式表的定义

为每个第三范式表定义一个数据库表

测量性能和需要的存储容量

修改先前设计的第三范式，以满足性能和存储需求

③ 面向对象数据库管理系统：

扩展的关系数据库途径：使用与关系数据库管理系统相同的方法

扩展的面向对象程序设计语言途径：不需要规范化属性的步骤

B：设计相应的服务

① 文件系统：

被存储的对象需要知道打开哪个文件，怎样把文件定位到正确的记录上，怎样检索出旧值，以及怎样用现有值更新它们

定义一个 ObjectServer 类，并创建它的实例

② 关系数据库管理系统：

被存储的对象，应该知道访问哪些数据库表，怎样访问所需要的行，怎样检索出旧值，以及怎样用现有值更新它们

定义一个 ObjectServer 类，并声明它的对象

③ 面向对象数据库管理系统：

扩展的关系数据库途径：使用与关系数据库管理系统相同的方法

扩展的面向对象程序设计语言途径：无须增加服务，只需给长期保存的对象加个标记，然后由面向对象数据库管理系统负责存储和恢复这类对象

二：设计类中的服务

（1）确定类中应有的服务

A：确定服务的总体思想

对象模型是进行对象设计的基本框架。必须把动态模型中对象的行为以及功能模型中的数据处理转换成由适当的类所提供的服务

动态模型中状态图中的状态转换执行对象服务的结果

功能模型指明了系统必须提供的服务

B：确定操作目标对象的启发规则

如果某个处理的功能是从输入流中抽取一个值，则该输入流就是目标对象

如果某个处理具有类型相同的输入流和输出流，而且输出流实质上是输入流的另一种形式，则该输入输出流就是目标对象

如果某个处理从多个输入流得出输出值，则该处理是输出类中定义的一个服务

如果某个处理把对输入流处理的结果输出给数据存储或动作对象，则该数据存储或动作对象就是目标对象

C：确定处理归属的启发规则

如果处理影响或修改了一个对象，则最好把该处理与处理的目标联系在一起

考察处理涉及的对象类及这些类之间的关联，从中找出处于中心地位的类

（2）设计实现服务的方法

A：设计实现服务的算法

算法复杂度：选用复杂度较低(效率较高)的算法，但不能过分追求高效率，应以能满足用户需求为准

容易理解与容易实现：容易理解与容易实现的要求往往与高效率有矛盾，设计者应该对这两个因素适当折衷

易修改：预测将来可能做的修改，并在设计时预先做些准备

B：选择数据结构

选择能够方便、有效地实现算法的物理数据结构

C：算法与数据结构的关系

① 分析问题寻找数据特点，提炼出所有可行有效的算法

② 定义与所提炼算法相关联的数据结构

③ 依据此数据结构进行算法的详细设计

④ 进行一定规模的实验与评测

⑤ 确定最佳设计

D：定义内部类和内部操作

增添一些用来存放在执行算法过程中所得出的中间结果的类，其需求陈述中没有提到。复杂操作往往可以用简单对象上的更低层操作来定义，因此，在分解高层操作时常常引入新的低层操作

第六节：设计关联和设计优化

一：设计关联

（1）关联

A：定义

在对象模型中，关联是联结不同对象的纽带，它指定了对象相互间的访问路径

B：确定实现关联的策略

选定一个全局性的策略统一实现所有关联

分别为每个关联选择具体的实现策略，以与它在应用系统中的使用方式相适应

（2）使用关联的方式

A：关联的遍历

单向遍历

双向遍历

B：实现单向遍历

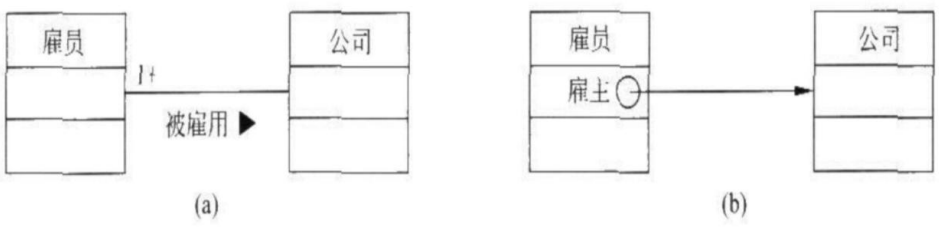


图 11-3　用指针实现单向关联

（a）关联；（b）实现

CSDN @t

若关联的重数是一元的，则实现关联的指针是一个简单指针

若关联的重数是多元的，则需要用一个指针集合实现关联

C：实现双向关联

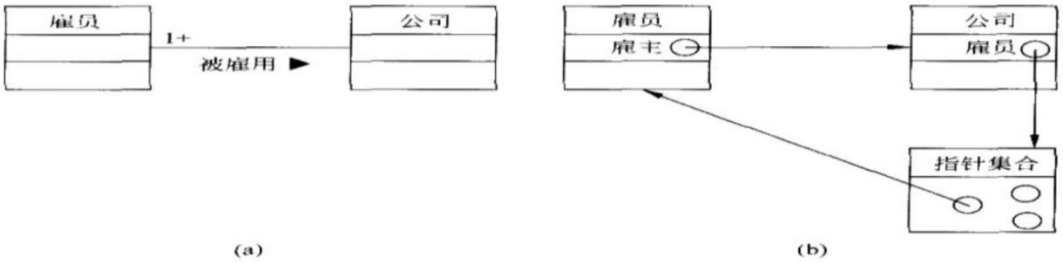


图 11-4　用指针实现双向关联

（a）关联；（b）实现

CSDN @快乐

只用属性实现一个方向的关联：如果两个方向遍历的频度相差很大，而且需要尽量减少存储开销和修改时的开销，则这是一种很有效的实现双向关联的方法

两个方向的关联都用属性实现：这种方法能实现快速访问。当访问次数远远多于修改次数时，这种实现方法很有效

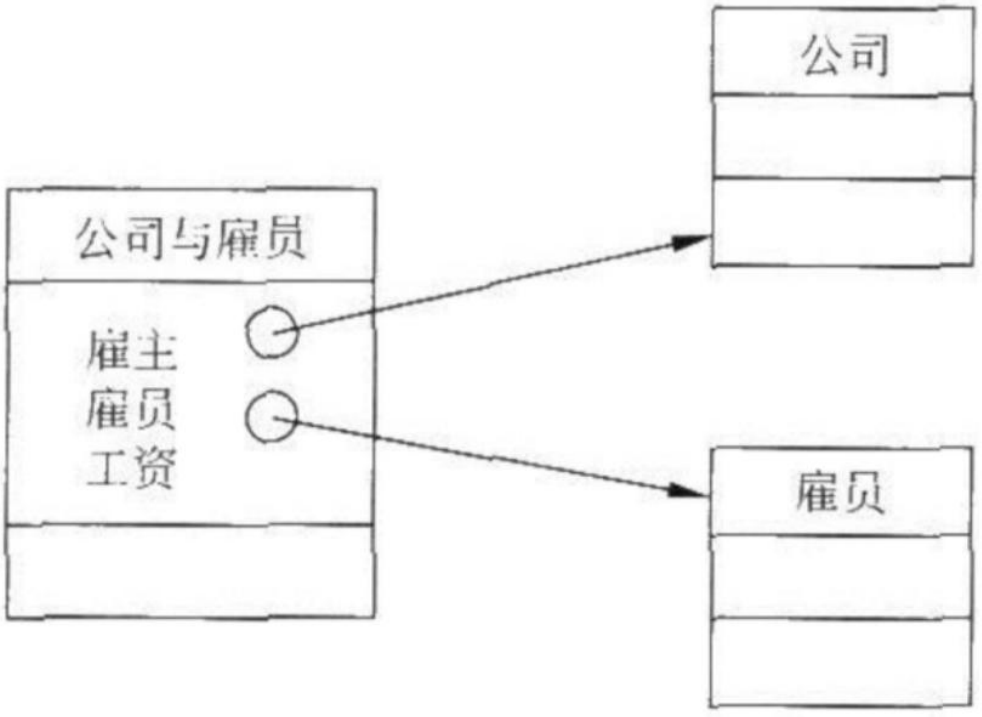


图 11-5　用对象实现关联

SDN @快乐江湖

用独立的关联对象实现双向关联：关联对象不属于相互关联的任何一个类，它是独立的关联类的实例

D：关联对象的实现

①：定义

用一个关联类来保存描述关联性质的信息，关联中的每个连接对应关联类的一个对象

②：方法

1) 对于一对一的关联，关联对象可以与参与关联的任一个对象合并
2) 对于一对多的关联，关联对象可以与多端对象合并
3) 对于多对多的关联，关联链的性质不可能只与一个参与关联的对象有关

二：设计优化

（1）确定优先级

A：必要性

系统的各项质量指标并不是同等重要的,必须确定各项质量指标的优先级,以便在优化设计时制定折中方案。系统的整体质量与制定的折中方案密切相关。最终产品成功与否，在很大程度上取决于是否选择好了系统目标

B：方法

在效率和清晰度之间寻求适当的折中方案。在折中方案中设置的优先级应当是模糊的

（2）提高效率的几项技术

① 增加冗余关联以提高访问效率
② 调整查询次序
③ 保留派生属性

（3）调整继承关系

A：继承关系

继承关系能够为一个类族定义一个协议，并能在类之间实现代码共享以减少冗余。一个基类和它的子孙类在一起称为一个类继承。在面向对象设计中,建立良好的类继承是非常重要的。利用类继承能够把若干个类组织成一个逻辑结构

B：建立类继承

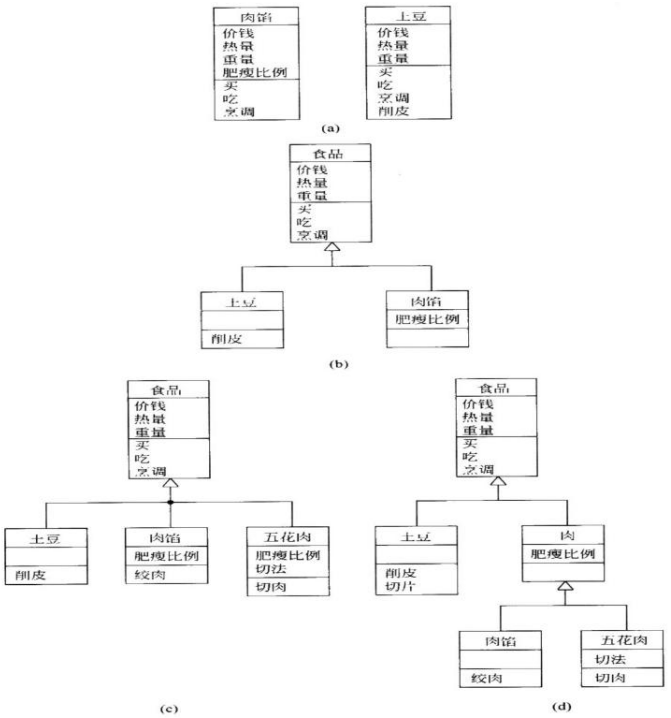


图 11-6 设计类继承的例子

（a）先创建一些具体类；（b）归纳出抽象类；

（c）进一步具体化；（d）再次归纳

①：抽象与具体

在设计类继承时，很少使用纯粹自顶向下的方法，通常的做法如下：

创建满足具体用途的类，然后对它们进行归纳

归纳出一些通用的类以后，根据需要再派生出具体类

进行一些具体化的工作后，再次归纳

②：为提高继承程度而修改类定义

在一组相似的类中存在公共的属性和公共的行为时，可以把这些公共的属性和行为抽取出来放在一个共同的祖先类中，供其子类继承，在对现有类进行归纳的时候，要注意下述两点：

不能违背领域知识和常识

应该确保现有类的协议不变

③：利用委托实现行为共享

仅当存在真实的一般一特殊关系(子类确实是父类的一种特殊形式)时，利用继承机制实现行为共享才是合理的

第十一章：面向对象实现

第一节：面向对象实现概述和程序设计语言

一：面向对象实现概述

（1）主要任务

把面向对象设计结果翻译成用某种程序语言书写的面向对象程序

测试并调试面向对象的程序

（2）面向对象程序质量的影响因素

面向对象设计的质量

采用的程序语言的特点

程序设计风格

（3）保证软件可靠性的方法

保证软件可靠性的主要措施是软件测试。面向对象测试的目标是用尽可能低的测试成本发现尽可能多的软件错误

二：程序设计语言

（1）面向对象语言的优点

A：面向对象设计结果的表达方式

面向对象语言：编译程序可以自动把面向对象概念映射到目标程序中

非面向对象语言：必须由程序员自己把面向对象概念映射到目标程序中

B：优点

① 一致的表示方法：面向对象开发基于不随时间变化的、一致的表示方法。既有利于在软件开发过程中始终使用统一的概念，也有利于维护人员理解软件的各种配置成分

② 可重用性：既可重用面向对象分析结果，也可重用相应的面向对象设计和面向对象程序设计结果

③ 可维护性：程序显式地表达问题域语义，对维护人员理解待维护的软件有很大帮助。在选择编程语言时，应该考虑的首要因素是哪个语言能最恰当地表达问题域语义

（2）面向对象语言的技术特点

在选择面向对象语言时应该着重考虑以下几种技术特点

A：支持类与对象概念的机制

①：内容

面向对象语言允许用户动态创建对象，并且可以用指针引用动态创建的对象。需要及时释放不再需要的对象所占用的内存

②：管理内存的方法

由语言的运行机制自动管理内存，即提供自动回收“垃圾”机制

由程序员编写释放内存的代码

B：实现聚集结构的机制

使用指针

使用独立的关联对象

C：实现泛化结构的机制

实现继承的机制

解决名字冲突的机制，即处理在多个基类中可能出现的重名问题

D：实现属性和服务的机制

实现属性的机制：

支持实例连接的机制

属性的可见性控制

对属性值的约束

实现服务的机制：

支持消息连接(表达对象交互关系)的机制

控制服务可见性的机制

动态联编(在发送消息前，无须知道接受消息的对象属于哪个类)

E：类型检查

①：分类

弱类型：语言仅要求每个变量或属性隶属于一个对象

强类型：语法规定每个变量或属性必须准确地属于某个特定的类

②：强类型语言的优点

有利于在编译时发现程序错误，提高软件的可靠性

增加了优化的可能性，提高软件的运行效率

③：适用性

使用强类型编译型语言开发软件产品，使用弱类型解释型语言快速开发原型

F：类库

存在类库，许多软构件就不必由程序员从头编写了，为实现软件重用带来很大方便。主要分为以下几类

① 包含实现通用数据结构的类，即包容类

② 实现各种关联的类

③ 独立于具体设备的接口类

④ 用于实现窗口系统的用户界面类

G：效率

提高效率的方法有

① 使用拥有完整类库的面向对象语言

② 优化查找继承树查找过程，从而实现高效率查找

H：持久保存对象

①：原因

为实现在不同程序之间传递数据，需要保存数据

为恢复被中断了的程序的运行，首先需要保存数据

②：方法

在类库中增加对象存储管理功能

使程序设计语言语法与对象存储管理语法无缝集成

J：参数化类

①：定义

参数化类是使用一个或多个类型去参数化一个类的机制，如果程序语言提供一种能抽象出这类共性的机制，则对减少冗余和提高可重用性是大有好处的

②：方法

定义一个参数化的类模板

把数据类型作为参数传递进来

K：开发环境

编辑程序

编译程序或解释程序(最重要)

浏览工具

调试器

（3）选择面向对象语言的标准

将来能否占主导地位

可重用性

类库和开发环境

其他因素

第二节：程序设计风格和测试策略

一：程序设计风格

（1）概念

A：良好的程序设计风格的重要性

能明显减少维护或扩充的开销

有助于在新项目中重用已有的程序代码

B：良好的面向对象程序设计风格内容

传统的程序设计风格准则

为适应面向对象方法所特有的概念而必须遵循的一些新准则

（2）提高可重用性

A：代码重用

内部重用：即本项目内的代码重用，主要是找出设计中相同或相似的部分，然后利用继承机制共享它们

外部重用：即新项目重用旧项目的代码，需要有长远眼光，反复考虑，精心设计

B：主要准则

①：提高方法的内聚

一个方法应该只完成单个功能。如果某个方法涉及两个或多个不相关

的功能，则应该把它分解成几个更小的方法

②：减小方法的规模

③：保持方法的一致性

保持方法的一致性，有助于实现代码重用，功能相似的方法应该有一致的名字、参数特征、返回值类型、使用条件及出错条件等

④：把策略与实现分开

策略方法：策略方法应该检查系统运行状态，并处理出错情况，它们并不直接完成计算或实现复杂的算法。其紧密依赖于具体应用

实现方法：实现方法仅仅针对具体数据完成特定处理，用于实现复杂的算法。在执行过程中发现错误，它们只返回执行状态而不对错误采取行动。其相对独立于具体应用

⑤：全面覆盖

输入条件的各种组合都可能出现，应针对所有组合写出方法

一个方法对空值、极限值及界外值等异常情况也应能够做出有意义的响应

⑥：尽量不使用全局信息

⑦：利用继承机制

调用子过程：把公共的代码分离出来，构成一个被其他方法调用的公用方法。可以在基类中定义这个公用方法，供派生类中的方法调用

分解因子：提高相似类代码可重用性的一个有效途径，是从不同类的相似方法中分解出不同的代码，把余下的代码作为公用方法中的公共代码，把分解出的因子作为名字相同算法不同的方法，放在不同类中定义，并被这个公用方法调用

使用委托：继承关系的存在意味着子类“即是”父类，因此，父类的所有方法和属性都应该适用于子类，仅当确实存在一般一特殊关系时，使用继承才是恰当的，当逻辑上不存在一-般一特殊关系时，为重用已有的代码，可以利用委托机制

把代码封装在类中：重用通过其他方法编写的、解决同一类应用问题的程序代码的一个比较安全的途径是把被重用的代码封装在类中

（3）提高可扩充性

主要准则有：

封装实现策略：应该把类的实现策略封装起来，对外只提供公有的接口，否则将降低今后修改数据结构或算法的自由度

不要用一个方法遍历多条关联链：一个方法应该只包含对象模型中的有限内容。违反这条准则将导致方法过分复杂，既不易理解，也不易修改扩充

避免使用多分支语句：可以利用 DO_CASE 语句测试对象的内部状态，而不要用来根据对象类型选择应有的行为，否则在增添新类时将不得不修改原有的代码

精心确定公有方法

（4）提高健壮性

主要准则有：

预防用户的错误操作

检查参数的合法性

不要预先确定限制条件

先测试后优化

二：测试策略

（1）经典的测试策略

测试软件的经典策略是，从“小型测试”开始，逐步过渡到“大型测试”。用软件测试的专业术语描述，可以分为以下三步：

单元测试

集成测试

确认测试、系统测试

（2）面向对象测试策略

A：面向对象的单元测试

最小的可测试单元是封装起来的类和对象。测试面向对象软件时，不能再孤立地测试单个操作，而应该把操作作为类的一部分来测试。在测试面向对象的软件时，传统的单元测试方法是不适用的，不能再在“真空”中(即孤立地)测试单个操作

B：面向对象的集成测试

①：策略

基于线程的测试：把响应系统的一个输入或一个事件所需要的那些类集成起来。分别集成并测试每个线程，同时应用回归测试以保证没有产生副作用

基于使用的测试：不使用服务器类的独立类，把独立类都测试完之后，再测试使用独立类的下一个层次的类(称为依赖类)。对依赖类的测试一层一层次地测试，直至把整个软件系统构造完为止

②：集群测试

集群测试是面向对象软件集成测试的一个步骤。在这个测试步骤中，用精心设计的测试用例检查一群相互协作的类，这些测试用例力图发现协作错误

C：面向对象的确认测试

在确认测试或系统测试层次，不再考虑类之间相互连接的细节。面向对象软件的确认测试也集中检查用户可见的动作和用户可识别的输出。为了导出确认测试用例,测试人员应该认真研究动态模型和描述系统行为的脚本，以确定最可能发现用户交互需求错误的情景

三：设计测试用例

（1）测试类的方法

A：随机测试

B：划分测试

①：优点

采用划分测试方法可以减少测试类时所需要的测试用例的数量

②：流程

把输入和输出分类

设计测试用例以测试划分出的每个类别

③：方法

基于状态的划分：根据类操作改变类状态的能力来划分类操作

基于属性的划分：根据类操作使用的属性来划分类操作

基于功能的划分：根据类操作所完成的功能来划分类操作

C：基于故障的测试

（2）集成测试方法

A：多类测试

①：随机测试

对每个客户类，使用类操作符列表来生成一系列随机测试序列

对所生成的每个消息，确定协作类和在服务器对象中的对应操作符

对服务器对象中的每个操作符，确定传递的消息

对每个消息，确定下一层被调用的操作符，并把这些操作符结合进测试序列中

②：划分测试

应该扩充测试序列以包括那些通过发送给协作类的消息而被调用的操作

根据与特定类的接口来划分类操作

B：从动态模型导出测试用例

类的状态图可以帮助人们导出测试该类的动态行为的测试用例。通过导出大量的测试用例，保证该类的所有行为都被适当地测试了。在类的行为导致与一个或多个类协作的情况下，应该使用多个状态图去跟踪系统的行为

第十二章：软件项目管理

软件项目管理： 通过计划、组织和控制等一系列活动，合理地配置和使用各种资源，以达到既定目标的过程

一：估算软件规模

（1）代码行技术

根据以往开发经验和开发数据，估算实现一个功能所需要的源代码行数

优点：

代码是所有项目都有的“产品”，容易计算代码行数

缺点：

源代码为软件配置的一个部分，用来衡量整个软件规模不太合理

不同语言实现同一软件所需要的代码行数不相同

不适用于非过程语言

（2）功能点技术

以功能点（FP）为单位度量软件规模

二：工作量估算

工作量是软件规模的函数，工作量的单位通常是人月(pm)

静态单变量模型（基本的 COCOMO 模型）

静态多变量模型（COCOMO2 模型）

动态多变量模型（putnam 模型）

三：进度计划

（1）甘特图(Cantt 图)

甘特图是制定进度计划的工具，优点是能形象描述任务分解情况，直观简洁和容易掌握

（2）工程网络

四：人员组织

软件开发的人员组织方式

民主制程序员组
主程序员组
现代程序员组

五：质量保证

软件质量就是软件与明确地和隐含地定义的需求相一致的程度

软件质量保障措施主要有

基于非执行测试（复审或评审）
基于执行测试（软件测试）
程序正确性的证明（数学方法）

六：软件配置管理

软件配置管理是在软件生命周期内管理变化的一组活动，用来标识、控制、报告变化，确保适当的实现了变化

基线：通过了正式复审的软件配置项，可以作为进一步开发的基础，只有通过正式的变化控制过程才能改变它

软件配置管理五项任务：

① 标识对象
② 版本控制
③ 变化控制
④ 配置审计
⑤ 状态报告

七：能力成熟度模型

能力成熟度模型是用于评价软件机构的软件过程能力成熟度模型，用于帮助软件开发机构建立一个有规模的，成熟的软件过程。五个等级从低到高为

① 初始级
② 可重复级
③ 已定义级
④ 已管理级
⑤ 优化级