Programming Methodology
程序设计方法学

第1章　程序设计方法简介

主讲：刘悦
yliu@staff.shu.edu.cn

主要内容

- 学科地位
- 程序设计语言和程序设计方法
- 程序设计方法产生与发展
- 程序设计方法学的定义与意义
- 结构化程序设计及其讨论的主要问题

内容线索

- 学科地位
- 程序设计语言和程序设计方法
- 程序设计方法产生与发展
- 程序设计方法学的定义与意义
- 结构化程序设计及其讨论的主要问题

计算机科学的分类与分支学科...

- 1、构造性数学基础
  - 数理逻辑、代数系统、图论、集合论等
- 2、计算的数学理论
  - 计算理论、高等逻辑、形式语言与自动机、形式语义等
- 3、计算机组成原理、器件与体系结构
  - 计算机原理与设计、体系结构等
- 4、计算机应用基础
  - 算法基础、程序设计、数据结构、数据库基础、微机原理与接口技术等

...计算机科学的分类与分支学科

- 5、计算机基本应用技术
  - 数值计算、图形学与图像处理、网络、多媒体、计算可视化与虚拟现实、人工智能等
- 6、软件基础
  - 高级语言、数据结构、程序设计、编译原理、数据库原理、操作系统原理、软件工程
- 7、新一代计算机体系结构与软件开发方法学
  - 并行与分布式计算机系统、智能计算机系统、软件开发方法学等

程序设计方法学是计算机科学中软件的一个方向

和其他课程之间的关系

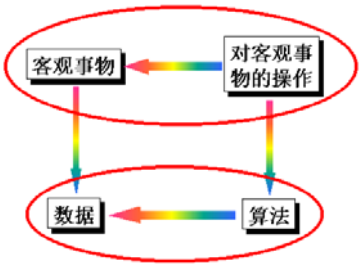
- 软件工程（管理）
- 数据结构与算法（实现）
- 程序设计语言学（运行）
- 程序设计技巧（技巧）
- 程序设计方法学（方法学）

内容线索

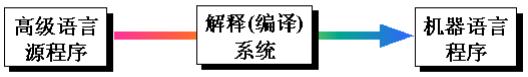
- ✓ 学科地位
- 程序设计语言和程序设计方法
- 程序设计方法产生与发展
- 程序设计方法学的定义与意义
- 结构化程序设计及其讨论的主要问题

程序设计与程序设计语言

- 程序设计就是用计算机语言把对数据进行处理的算法表达出来

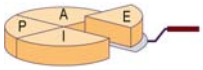


- 程序的表达手段就是程序设计语言



程序设计方法的发展

- 面向计算机的程序设计
  - 机器语言，注重机器，逐一执行
- 面向过程的程序设计
  - 结构化程序设计 —— 以数据为中心
  - 自顶向下逐步求精
  - 按功能分解，数据和操作分离
- 面向对象程序设计 —— 以对象为中
- 心注重对象，抽象成类
  - 对象、类(Abstract）、封装(Encapsulation)、
  - 继承(Inheritance)、多态性(Polymorphism)、



- 组件程序设计 —— 以组件为中心
  - 即插即用
- 其他程序设计
  - 递归程序设计、嵌入式程序设计

综合使用

程序设计思想的进步…

- 1967年5月20日，在挪威奥斯陆郊外的小镇莉沙布举行的iFiP TC-2 工作会议上，挪威科学家Ole-Johan Dahl和Kristen Nygaard正式发布了Simula 67语言。Simula 67被认为是最早的面向对象程序设计语言，它引入了所有后来面向对象程序设计语言所遵循的基础概念：对象、类、继承。之后，在1968年2月形成了Simula 67的正式文本。


Ole-Johan Dahl (1978)

- 也是在1968年，荷兰教授E.W.Dijkstra提出了“GOTO语句是有害的”观点，指出程序的质量与程序中所包含的GOTO语句的数量成反比，认为应该在一切高级语言中取消GOTO语句。这一观点在计算机学术界激起了强烈的反响，引发了一场长达数年的广泛的论战，其直接结果是结构化程序设计方法的产生。


Kristen Nygaard (1978)

…程序设计思想的进步…

- 在20世纪60年代，软件曾出现过严重危机，由软件错误而引起的信息丢失、系统报废事件屡有发生。为此，1968年，荷兰学者E.W.Dijkstra提出了程序设计中常用的GOTO语句的三大危害：破坏了程序的静动一致性；程序不易测试；限制了代码优化。此举引起了软件界长达数年的论战，并由此产生了结构化程序设计方法，同时诞生了基于这一设计方法的程序设计语言Pascal。
- 由瑞士计算机科学家Niklaus Wirth开发的Pascal，一经推出，它的简洁明了以及丰富的数据结构和控制结构，为程序员提供了极大的方便性与灵活性，同时它特别适合微计算机系统，因此大受欢迎。结构化程序设计思想采用了模块分解与功能抽象和自顶向下、分而治之的方法，从而有效地将一个较复杂的程序系统设计任务分解成许多易于控制和处理的子程序，便于开发和维护。因此，结构化方法迅速走红，并在整个20世纪70年代的软件开发中占绝对统治地位。

…程序设计思想的进步…

- 到了70年代末期，随着计算机科学的发展和应用领域的不断扩大，对计算机技术的要求越来越高。结构化程序设计语言和结构化分析与设计已无法满足用户需求的变化，于是面向对象技术开始浮出水面。
- 面向对象程序设计方法起源于Simula 67语言。在程序设计语言的发展史上，20世纪60年代后期是承上启下的重要时期。这一时期有三种重要的语言问世，即Simula 67、由一批顶尖计算机科学家共同设计的Algol 68、以及为iBM 360系列机配套开发的PL/i。这三种语言虽均有所创新，但Simula 67的面向对象概念的影响是最巨大而深远的。它本身虽因为比较难学、难用而未能广泛流行，但在它的影响下所产生的面向对象技术却迅速传播开来，并在全世界掀起了一股OO热潮，至今盛行不衰。面向对象程序设计在软件开发领域引起了大的变革，极大地提高了软件开发的效率，为解决软件危机带来了一线光明。

…程序设计思想的进步

- 事实表明，面向对象程序设计方法虽然比结构化方法能更自然地表现现实世界，但它不是灵丹妙药，并不能解决所有问题，它本身存在固有的内在的局限性。
- 面向方面编程（AOP）正是为了改进上述程序设计方法学的不足。AOP被视为是“后”面向对象时代的一种新的重要的程序设计技术。
- Web 服务通过允许应用程序以对象模型中立的方式实现互连，从而提供了一个更强大、更灵活的编程模型，并将对软件开发方法产生巨大的影响。

软件开发语言龙虎斗…

- 从1952年第一个高级语言Short Code诞生到现在，程序设计语言先后出现了不同类型、不同版本不下数百种语言。

表　部分程序设计语言诞生时间

语言及版本	发布者	时间
BASIC	Dartmouth学院	1964年
BASIC解释器	Microsoft公司	1975年
Quick BASIC 1.0	Microsoft公司	1985 年
Turbo Basic 1.0	Borland公司	1987 年
Visual Basic 1.0	Microsoft公司	1991 年
Visual Basic 6.0	Microsoft公司	1998 年
Visual Basic.NET	Microsoft公司	2001年
Turbo Pascal 1.0	Borland公司	1983年
Delphi 1.0	Borland公司	1995年
Delphi 8 for .NET	Borland公司	2003年
Turbo C	Borland公司	1989年
Turbo C++ 1.01	Borland公司	1991年
Borland C/C++3.0	Borland公司	1992年
Visual C++ 1.0	Microsoft公司	1995年
Visual C++ 6.0	Microsoft公司	1998年
Visual C#.NET	Microsoft公司	2001年
C++ BuilderX	Borland公司	2003年
Java	Sun公司	1996年
Jbuilder 1.0	Borland公司	1997年
Java2	Sun公司	1998年
JbuilderX	Borland公司	2003年

…软件开发语言龙虎斗…

- 1964年诞生的Basic语言是较早出现且至今仍有较大影响的语言之一。1975年，微软以Basic解释器创业，尽管1987年Borland 公司成功地推出了Turbo Basic 1.0，但由于其内部原因而放弃了Basic市场，从而使微软在Basic领域是一览众山小。1991年，Visual Basic 1.0问世，它允许程序员在一个所见即所得的图形界面中迅速完成开发任务。1998年发布的Visual Basic 6.0是传统Visual Basic中功能最全、应用最广的一个版本。伴随着.NET平台的横空出世，Visual Basic.NET又以一个全新的面目出现在我们面前。
- 尽管Basic语言诞生较早，但其真正商业化是从1983年开始的。而在这一年，Borland公司又推出了著名的Pascal编译器Turbo Pascal 1.0，在一定程度上抢了Basic的风头。Turbo Pascal 1.0正式开创了Borland影响PC开发工具十几年的历史。尽管微软也曾经推出了Microsoft Pascal，但无疾而终。随着Turbo Pascal 第9版的推出，Pascal语言也得到了不断的发展。1995年，Borland发布了Delphi，使Pascal成为一种可视化编程语言。Delphi 1.0号称“VB Killer”，又一次在危难的时刻拯救了Borland。

…软件开发语言龙虎斗…

- 微软和Borland之间的竞争并不只是在Basic和Pascal方面，在C语言世界里也同样如此。早期的C/C++开发环境是Borland于1989年5月发布的Turbo C。同一时期，微软也推出了Microsoft C/C++，不过它直到6.0版，市场反映都一直平平。之后，Borland C/C++ 3.0问世，开启了Borland雄霸C/C++编译器常达五六年之久的序幕。尤其是其3.1版的畅销，使Borland一举击溃了Microsoft C/C++，市场占有率超过了50%。但后来由于Borland决策上的一些失误，随后的4.0和4.5版市场反映极差，相反微软于1995年成功地推出了Visual C++ 1.0，并获得市场好评，它不但在编译器方面能与Borland C/C++ 3.1相抗衡，在整合发展环境方面更加领先。应该说，这是两家公司C/C++战役的转折点，也是C/C++发展的里程碑。在此之后，Visual C++一路高歌猛进到6.0版，并最终发展为.NET时代的C#。不过，Borland也并没有一蹶不振，2003年5月，Borland针对.NET发布了C#Builder，之后又发布了C++ BuilderX。

…软件开发语言龙虎斗

- 说到编程语言，不能不说Java。Java是internet时代的产物，同其他前辈语言相比，其最大的特色在于“一次编码，多处运行”。Sun公司是Java语言的缔造者，但对Java开发环境做出最大贡献的是Borland的JBuilder系列产品。1997年，Borland发布JBuilder 1.0，进军Java跨平台开发。这之后，微软曾试图染指Java开发环境市场，并影响Java的技术发展，并在Visual Studio系列中，将Visual J++从1.0直接跨越到6.0。但是，JBuilder至今仍控制着Java的开发环境市场。

各种语言

- 你学的第一门编程语言是什么？_C
- 适合于入门的脚本编程语言
  - 蔡学镛先生的《编程ING》选择了REBOL编程语言
- Python：认识编程是怎么回事，训练基本编程技能
  - http://www.python.org
- MATLAB和Scilab：训练算法的设计与编程实现能力
- Office+VBA：用VBA代码控制Office，让各种工作自动化
- Processing编程语言：体会图形与动画的魅力
  - http://www.processing.org/
- Small Basic：适合“零编程基础”人的编程语言
  - http://www.smallbasic.com/
- HTML 5 + JavaScript：互联网时代的主流编程语言
- 以编译型的语言作为入门级编程语言
  - Java：“人多势众”的主流面向对象编程语言
  - C#：面向对象编程语言的集大成者

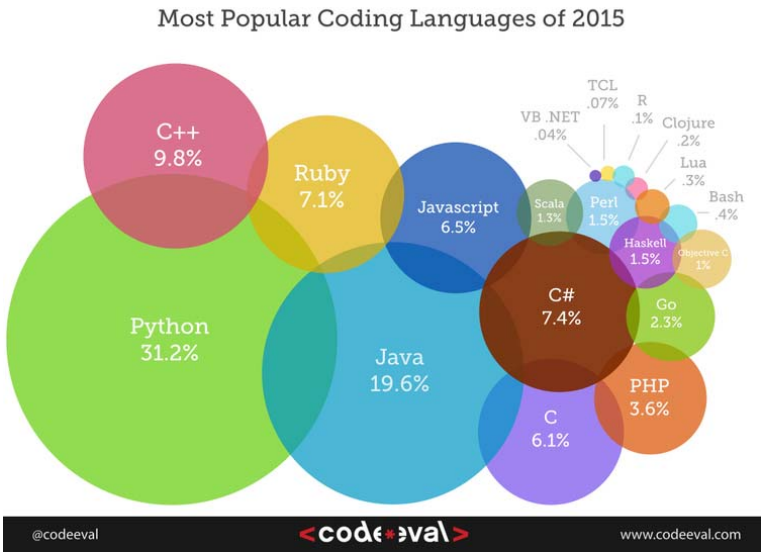
历年年度语言

- 2003年： C++
- 2004年： PHP
- 2005年： Java
- 2006年： Ruby
- 2007年： Python
- 2008年： C
- 2009年： Go
- 2010年： Python
- 2011年： Objective-C
- 2012年： Objective-C [4]
- 2013年： Transact-SQL

历年最热门编程语言

2015 Rank		Change	2014 Rank	Change	2013 Rank	Change	2012 Rank
1	Python	0	1	0	1	0	1
2	Java	0	2	0	2	0	2
3	C++	0	3	0	3	0	3
5	Ruby	-1	4	0	4	0	4
4	C#	2	6	2	8	1	9
7	C	0	7	-1	6	4	10
6	JavaScript	-1	5	2	7	-1	6
8	PHP	0	8	-3	5	0	5
9	Go	1	10	-1	9	-2	7
10	Perl	-1	9				
11	Haskell	0	11				
12	Scala	0	12	-1	11	0	11
13	Objective-C	0	13	-1	12	1	13
14	Bash	1	15				
15	Lua	1	16				
16	Clojure	-2	14	-4	10	-2	8
17	R*						
18	Tcl	-1	17	-4	13	-1	12
19	Visual Basic.NET*						

2015 年最热门编程语言排行榜



2012年6月编程语言排行

Position Jun 2012	Position Jun 2011	Delta in Position	Programming Language	Ratings Jun 2012	Delta Jun 2011	Status
1	2	↑	C	17.725%	+1.45%	A
2	1	↓	Java	16.265%	-2.32%	A
3	3	=	C++	9.358%	-0.47%	A
4	7	↑↑↑	Objective-C	9.094%	+4.66%	A
5	4	↓	C#	7.026%	+0.18%	A
6	6	=	(Visual) Basic	6.047%	+1.32%	A
7	5	↓↓	PHP	5.287%	-1.31%	A
8	8	=	Python	3.848%	-0.05%	A
9	9	=	Perl	2.221%	-0.09%	A
10	12	↑↑	Ruby	1.683%	+0.20%	A
11	11	=	JavaScript	1.474%	-0.03%	A
12	29	↑↑↑↑↑↑↑↑↑	Visual Basic .NET	1.216%	+0.78%	A
13	13	=	Delphi/Object Pascal	1.150%	+0.08%	A
14	14	=	Lisp	0.986%	+0.05%	A
15	21	↑↑↑↑↑↑	Logo	0.860%	+0.31%	A-
16	15	↓	Pascal	0.844%	+0.11%	A
17	17	=	Transact-SQL	0.705%	+0.05%	A
18	19	↑	Ada	0.681%	+0.08%	B
19	22	↑↑↑	PL/SQL	0.637%	+0.13%	A-
20	10	↓↓↓↓↓↓↓↓↓↓	Lua	0.635%	-1.40%	B

2015年11月编程语言排行(1)

Nov 2015	Nov 2014	Change	Programming Language	Ratings	Change
1	2	↑	Java	20.403%	+6.01%
2	1	↓	C	17.145%	-0.32%
3	4	↑	C++	6.198%	+0.10%
4	5	↑	C#	4.318%	-0.67%
5	7	↑	Python	3.771%	+1.18%
6	6		PHP	3.248%	+0.20%
7	8	↑	JavaScript	2.473%	+0.38%
8	10	↑	Visual Basic .NET	2.223%	+0.16%
9	14	⇈	Ruby	2.038%	+0.83%
10	9	↓	Perl	2.032%	-0.04%

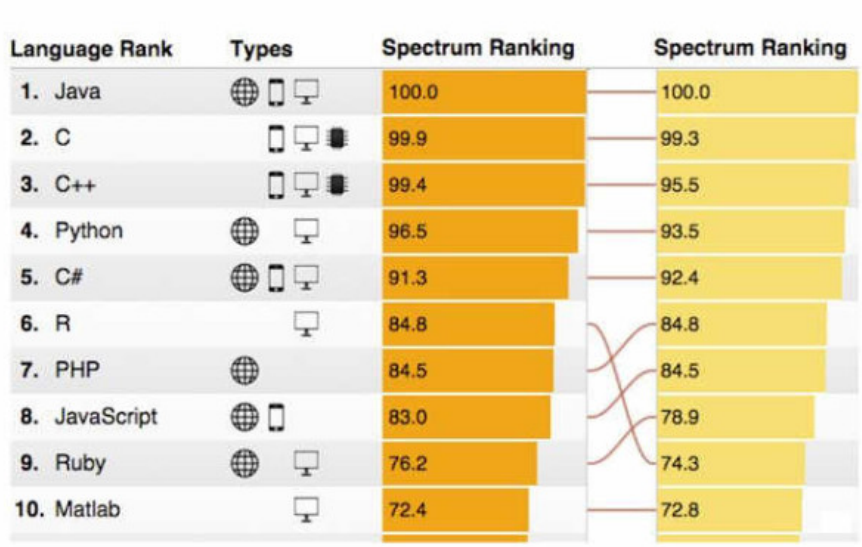
1 - 23

2015年11月编程语言排行(2)

Nov 2015	Nov 2014	Change	Programming Language	Ratings	Change
11	29	⇈	Assembly language	1.883%	+1.28%
12	15	↑	Delphi/Object Pascal	1.682%	+0.73%
13	11	↓	Visual Basic	1.681%	+0.02%
14	3	⇊	Objective-C	1.426%	-7.64%
15	18	↑	Swift	1.236%	+0.40%
16	24	⇈	MATLAB	1.185%	+0.43%
17	19	↑	Pascal	1.099%	+0.27%
18	17	↓	PL/SQL	1.032%	+0.16%
19	12	⇊	R	1.013%	-0.53%
20	28	⇈	COBOL	0.921%	+0.32%

1 - 24

IEEE Spectrum与计算机记者Nick Diakopoulos携手给出的编程语言人气排行榜



Language Rank	Types	Spectrum Ranking	Spectrum Ranking
1. Java		100.0	100.0
2. C		99.9	99.3
3. C++		99.4	95.5
4. Python		96.5	93.5
5. C#		91.3	92.4
6. R		84.8	84.8
7. PHP		84.5	84.5
8. JavaScript		83.0	78.9
9. Ruby		76.2	74.3
10. Matlab		72.4	72.8

云计算时代最具潜力的12种编程语言

内容线索

- ✓ 学科地位
- ✓ 程序设计语言和程序设计方法
- 程序设计方法产生与发展
  - 产生背景
  - 发展过程
- 程序设计方法学的定义与意义
- 结构化程序设计及其讨论的主要问题

背景

- 程序设计的发展过程
  - 手工艺式的设计方法→工程化的设计方法
    （50年代后—60年代初）（本世纪60年代以来）
- 软件危机
  - 对成本、进度的估算难以准确
  - 用户对已完成的软件系统常常不满意
  - 软件产品质量不可靠
  - 软件常常难以维护
  - 软件成本的上升
  - 缺少文档资料
  - 软件生产速度跟不上实际需要

发展过程

- 不同程序设计语言的出现带来不同的程序设计方法
  - 1） **五十年代：** 机器指令代码、手工艺劳动，少、快、省
  - 2）**六十——七十年代初期：**第一个高级语言FORTRAN语言出现，简化程序设计，摆脱机器特性限制，集精力于算法本身

问题：

➢1966年，Bohm,Jacopini证明：只要三种控制结构就能表达用一个入口、一个出口的框图（流程图）所能表达的任何程序逻辑

➢1968年 Dijkstra 《Communication of ACM》“GOTO有害论” 打响了第一炮

➢NATO 在德国软件工程会议

建议：GOTO语句太易把程序弄乱，应从一切高级语言中去掉；只用三种基本控制结构就可以写各种程序，而这样的程序可以自顶向下阅读而不会返回。具有历史意义的引起激烈讨论，提出一种新的程序设计思想、方法和风格——结构化程序设计思想与概念，以期提高软件生产率和降低软件维护代价

➢七十年代初期，大型系统软件，操作系统，数据库出现，给程序设计带来新的问题：“软件危机”

…程序设计方法学的产生和发展

- 七十年代——现在： 结构化程序设计——程序正确性证明的研究
  - 1972年，Mills进一步提出程序应该只有一个入口和出口，补充了结构程序的规则。
  - 1974年，D.E.Knuth 对GOTO争论进行了总结：“有些情形，主张废除；另一些情况，主张引进”。
  - 1980年，Gries综合了以谓词演算为基础的证明系统，称为“程序设计科学”。首次把程序设计从经验、技术上升为“科学”。
- 今天：
  - 方法学的研究获得了不少成果，除结构化程序设计日益完善外，模块化程序设计、递归程序设计、逐步求精方法等均成为有效的方法，抽象数据的代数规范，程序的形式化推导技术正在发展，特别是程序变换技术和自动化方面虽不成熟，已取得可喜进展。

图灵奖获得者

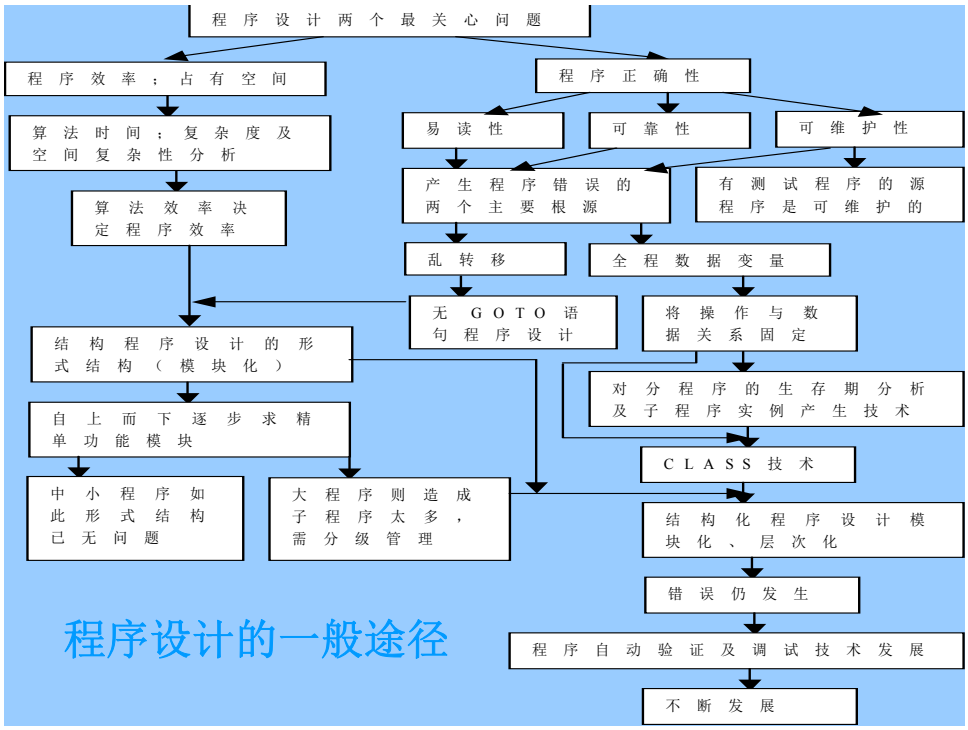
1972年	Edsgar Wybe Dijkstra	最先察觉“Goto有害”的计算机科学大师
1974年	Donald Ervin Knuth	经典巨著《计算机程序设计的艺术》的年轻作者
1978年	Robert W. Floyd	前后断言法的创始人，归纳断言法
1980年	Charles Antony Richard Hoare	从QuikSort、CASE到程序设计语言公理化
1984年	Niklaus Wirth	PASCAL之父及结构化程序设计的首创者

内容线索

- ✓ 学科地位
- ✓ 程序设计方法产生与发展
- ■ 程序设计方法学的定义与意义
- ■ 结构化程序设计及其讨论的主要问题

程序设计方法学

- ■ 讨论程序的性质以及程序设计的理论和方法的一门学科
  - ❖ 包括：
    - ◆ 结构程序设计
    - ◆ 数据抽象与模块化程序设计
    - ◆ 程序正确性证明
    - ◆ 程序变换
    - ◆ 程序的形式证明与推导
    - ◆ 程序综合技术
    - ◆ 面向对象的程序设计
    - ◆ 等等



内容线索

- ✓学科地位
- ✓程序设计方法产生与发展
- ✓程序设计方法学的定义与意义
- ■结构化程序设计及其讨论的主要问题

结构化程序设计及其讨论的主要问题

- 什么是结构程序设计
  - 结构程序设计是避免用GOTO语句的一种程序设计
  - 结构程序设计是自顶向下的程序设计
  - 结构程序设计是一种组织和编制程序的方法,利用它编制的程序是容易理解和容易修改的
  - 程序结构化的一个主要功能是使得正确性的证明容易实现
  - 结构程序设计允许在设计过程中的每一步验证其正确性,即自动导致自我说明与自我捍卫的程序风格
  - 结构程序设计讨论了如何将任何大规模的和复杂的流程图转换成一种标准的形式,使得它们能够用几种标准的控制结构(通常是顺序、分支和重复)通过重复和嵌套来表示

结构程序设计

- 结构程序设计是一种进行程序设计的原则和方法，按照这种原则和方法设计出的程序的特点是
  - 结构清晰
  - 容易阅读
  - 容易修改
  - 容易验证
- 按照结构程序设计的要求设计出的程序设计语言称为结构程序设计语言
- 利用程序设计语言或者说按照结构程序设计的思想编制出的程序称为结构化程序，或者良结构的程序
- 结构化程序
  - 顺序、分支和循环三种基本控制结构和程序块只有“一个入口和一个出口”的原则

结构化程序讨论的主要问题

- （1） goto语句
- （2）程序的结构
- （3）逐步求精程序设计
- （4）自顶向下的设计、编码和调试
- （5）主程序员组的组织形式

关于GOTO语句…

- 1、用“允许使用GOTO语句”的高级语言设计程序尽量避免使用GOTO语句
- 2、语言提供了多种控制结构，为避免使用GOTO语句创造条件
- 3、消除GOTO 语句的通常方法有
  - 增加辅助变量
  - 改变程序执行顺序

…关于GOTO语句…

- 例1

引入逻辑变量p

```
L1: if B1 then go to L2 fi;
    s1;
    if B2 then go to L2 fi;
    s2;
    go to L1;
L2: s3;
```

```
p:=true;
while p do
  if B1 then p:=false;
        else s1; if B2 then p:=false;
          else s2 fi fi;
s3;
```


…关于GOTO语句…

- 例2：查表程序
  - 在一个表中有m个不同的数A[1], A[2],…, A[m],在该表中查找数x，若找到则打印，否则将该数添加到表中

引入逻辑变量p

```
…
for i:=1 to m do
  if A[i]=x then go to 1 fi;
  m:=m+1; A[m]:=x; go to 2;
1: wirte(i,x);
2: …
```

```
…
p:=false;
for i:=1 to m do
  if A[i]=x then p:=true;y:=i; fi;
if p then wirte(y,x);
  else m:=m+1; A[m]:=x; fi;
…
```

…关于GOTO语句…

- 例2

改变程序执行顺序

```
…
for i:=1 to m do
  if A[i]=x then go to 1 fi;
  m:=m+1; A[m]:=x; go to 2;
1: wirte(i,x);
2: …
```

```
…
i:=1;
while (A[i] ≠x) ∧(i<m) do i:=i+1;
if i>m then m:=m+1; A[m]:=x;
        else write(i,x) fi;
…
```


C++中的Goto语句…

- Goto语句，实现异常处理编程，最初也最原始的支持手段

```
void main(int argc, char* argv[]){
  if (Call_Func1(in, param out) {
    // 函数调用成功，我们正常的处理
    if (Call_Func2(in, param out){
      // 函数调用成功，我们正常的处理
      while(condition){
        //do other job
        // 如果错误直接跳转
        if (has error) goto Error;
          //do other job
        }
      }
    // 如果错误直接跳转
    else goto Error;
  }
  // 如果错误直接跳转
  else goto Error;
    // 错误处理模块
    Error:
    process_error();
exit();}
```

…C++中的Goto语句

- 为什么不建议使用goto语句来实现异常处理编程

虽然goto 语句能有效地支持异常处理编程的实现。但是没有人建议使用它，即便是在C语言中。因为

（1） goto语句能破坏程序的结构化设计，使代码难于测试，且包含大量goto的代码模块不易理解和阅读。它一直遭结构化程序设计思想所抛弃，强烈建议程序员不易使用它

（2） 与C++语言中提供的异常处理编程模型相比，它的确是太弱了一些

- 例如，它一般只能是在某个函数的局部作用域内跳转，也即它不能有效和方便地实现程序控制流的跨函数远程的跳转

（3） 如果在C++语言中，用goto语句来实现异常处理，那么它将给面向对象构成极大破坏，并影响到效率

Java对Goto语句的处理

- Java不用goto关键词，将“GOTO”作为保留字，但是并不支持“GOTO”语句
- 在C#中，goto允许你转到指定的标签。不过，C#以特别谨慎的态度对待goto，比如它不允许goto转入到语句块的内部
- 在Java中，可以用带标签的语句加上break或continue取代C#中的goto

Delphi-object pascal对Goto语句的处理...

- 支持
- 使用goto 语句前要先声明标号，然后在要跳转到的语句的前面标上标号，最后才能使用goto 语句跳转到指定标号所标记的那个语句
  - label: mygoto;
  - ...
  - GOTO mygoto;
- 下面是利用goto 语句实现计数循环来完成阶乘运算的一个例子：

Delphi对Goto语句的处理...

- 使用GOTO语句要注意：标号所在的语句必须与GOTO语句在同一个块内
- 建议尽量不要使用GOTO语 句，这样很容易破坏结构化的程序

```
function fac(n:integer):real;
label 10,20; // 声明两个语句标号
var
 i:integer;
 r:real;
begin
 r:=1;
 i:=2;
10:
 if i<=n then
  r:=r*i
 else
  goto 20;
 inc(i);
 goto 10;
20:
 result:=r
end;
```

VB对Goto语句的处理...

- GoTo line
  - GoTo 只能跳到它所在过程中的行
  - 尽可能使用结构化控制语句(Do...Loop、For...Next、if...Then...Else、Select Case)
- GoTo 语句示例
  - 本示例使用 GoTo 语句在一个过程内的不同程序段间作流程控制，不同程序段用不同的“程序标签”来区隔
- goto语句在在vb.net已经不再被支持

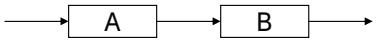
... VB对Goto语句的处理

```
Sub GotoStatementDemo()
   Dim Number, MyString
    Number = 1 ‘ 设置变量初始值。
    ’ 判断 Number 的值以决定要完成哪一个程序区段（以“程序标签”来表式）。
    if Number = 1 Then GoTo Line1 Else GoTo Line2

   Line1:
    MyString = "Number equals 1"
    GoTo LastLine ' 完成最后一行。
   Line2:
   ' 下列的语句根本不会被完成。
    MyString = "Number equals 2"
   LastLine:
    Debug.Print MyString ' 将“"Number equals 1""显示在“立即”窗口。
   End Sub
```

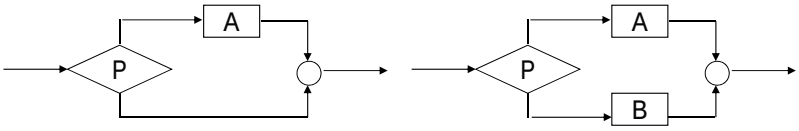
程序的结构...

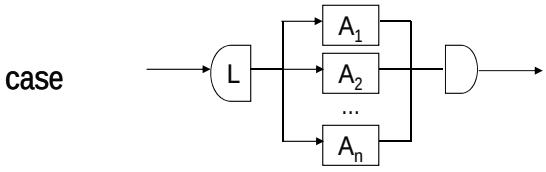
- 好结构
  - 结构清晰、易于读写、易于验证、可靠性好、可维护性高的程序
- 结构程序
  - 只包含三种基本控制结构，且程序块均只有一个入口和一个出口
  - 序列结构（开型结构）



...程序的结构...

- 选择结构（开型结构）

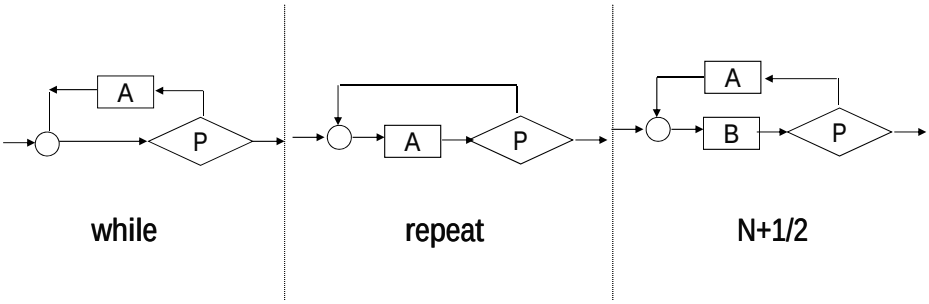




开型结构：
没有出现将控制转移到本结构入口的情形

...程序的结构

- 循环结构（闭型结构）



闭型结构：
在一定的条件下，将控制转移到本结构的入口处

逐步求精的程序设计方法简介...

- Wirth

我们对付复杂问题的最重要的办法是抽象。
所以对于一个复杂问题，不应马上用计算机指令、数字与逻辑字来表达，而应该用较为自然的抽象语句来表示，从而得出抽象程序

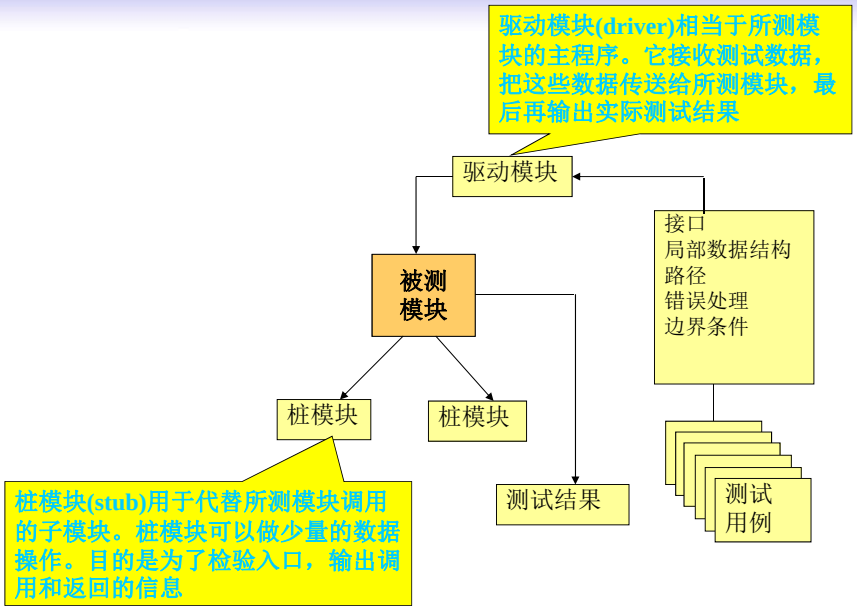
- 抽象程序对抽象的数据进行某些特定的运算，并用某些合适的记号（可能是自然语言）来表示
- 对抽象程序作进一步的分解，并进入下一层的抽象
- 这样，精细化过程一直进行下去，直到程序能被计算机接受为止。此时的程序可能是用某种高级语言或及其指令书写的

…逐步求精的程序设计方法简介

- 逐步求精的方法
  - 在编制一个程序时，首先考虑程序的整体结构而忽视一些细节问题，然后逐步地、一层一层地细化程序直至用所选的语言完全描述每一个细节，即得到所期望的程序为止

自顶向下的设计、编码和调试

- Refine是在一个模块内进行的
  - 先从该模块的功能描述出发，一层层地逐步细化，直到最后分解、细化成高级语言语句为止
  - 一个大而完成的程序设计分解细化成一个个程序模块，每个模块一般是几十行→上百行，框图只占一页。程序只有一个入口、一个出口，只用了3~6种基本控制结构，所完成功能较明确独立。
  - 这样，模块易于阅读理解，也便于检查其正确性。
  - 在这种分解中，主程序引用下层模块或一个模块引用再下一层模块，都通过“子程序调用”或“空功能”实现。
- 调试：“假模块”（dummy module）（驱动模块）,存根（sub）（桩模块）
  - 上层模块调用下层模块，编一个很简单的模块作为代用



例1...

- 编写一个程序，打印出前N个素数
  - 第一步：根据所提出问题，程序的结构可以写成

```
Void EX1(){
  int i, x;
    x=1;
    for (i=1;i<=N;i++){
        x="下一个素数";
        write(x);
    }
}
```

…例1…

- 第二步：对语句“x:=“下一个素数”;”进一步求精

```
Void EX1(){
  int i, x;
  x=1;
    for( i=1; i<=N;i++)
    {
        x=“下一个素数”;
        write(x);
     }
}
```

2

```
Do{x=x+1;
  Prim==“x是一个素数”;
}while(!prim);
```

3

…例1…

- 第三步：对语句“prim:=“x是一个素数”;”进一步求精

```
Do{x=x+1;
  Prim==“x是一个素数”;
}while(!prim);
```

3

```
K=2;prim=true;
While(prim && k<=lim)
Do{
     prim=“x不能被k整除”;
     k=k+1;
};
```

4

…例1 …

- 第四步：对语句“prim:=“x不能被k整除” ;”进一步求精
  - 原始地，素数是只能被1及它自身除尽，而不能被2，3，…，x-1整除
  - 问题：效率不高，需要改进

```
K=2;prim=true;
While( prim && k<=lim)
Do{
    prim=“x不能被k整除”;
    k=k+1;
 };
```

4

```
prim=(x % k) ≠ 0;
```

5

lim=x-1

…例1 …

- 第五步：进一步加工
  - 引入数论知识：
    - （1）lim→√x
    - （2）2单独处理
  - 只要检查x能不能被不超过√x的素数整除即可

5

```
x=x+2;lim=sqrt(x);
K=2;prim=true;
While( prim AND k<=lim)
do{
    prim=(x % k) ≠ 0;
    k=k+1;
 }
```

lim= x*x

…例1 …

- 第六步：再进一步加工
  - 筛选法
  - x能被k整除，则必定为k的素因子整除，所以确定x是不是素数，则只要检查x能不能被不超过√x的素数整除即可
  - 设数组p，p[k]中存放第k个素数


```
do{
  x=x+2; K=2;prim=true;
  While( prim AND k<=lim) do
    {
      prim=(x % p[k]) ≠ 0;
      k=k+1;
    }
}while( !prim);
p[i]=x;
```

p[lim]>√x而p[lim-1] ≤√x

或者

$(p[lim])^2 > x$而$(p[lim-1])^2 \le x$

这样，对所有满足k<lim的k，均有： p[k] ≤√x

…例1

- 综上所述，得到求解这个问题的一个完整的程序

```
#include <iostream>
#include <string>
#include <math.h>
using namespace std;
int main(){
   int N;
   int x;
   int i,k,lim;
   bool prim = false;
   cin >> N;
   int *p = new int[N+1];
   p[1] = 2;
   cout << p[1]<<endl;
   x = 1; lim = 1;
   for(i = 2; i <= N; i++){
      do{
         x += 2;
         if( lim * lim < x)
            lim += 1;
         k = 2;
         prim = true;
         while(prim && k < lim){
            prim = (x % p[k]) != 0;
            k +=1;}
      }
      while(!prim);
      p[i] = x;
      cout << x << endl;
   }
   delete[] p;
   return 0;
}
```

例2

- 例，用筛选法求100以内的素数
  - 筛选法就是从2到100中去掉2,3,5,7的倍数，剩下的就是100以内的素数。
  - 为了解决这个问题，可先按程序功能写出一个框架。


```
main ( ) {          //程序框架
   建立2到100的数组A[ ]，其中A[i]＝i；
      ----------------------------------1
   建立2到10的素数表 B[ ]，其中存放2
      到10以内的素数；----------2
   若A[i]＝i是B[ ]中任一数的倍数，则
      剔除A[i]；----------------------3
   输出A[ ]中所有没有被剔除的数；
      ----------------------------------4
}
```

```
main ( ) {
  /*建立2到100的数组A[ ]，其中A[i]＝i*/
  for ( i = 2； i <= 100； i++ ) A[i] = i；
  /* 建立2到10的素数表B[ ]，其中存放2到10
  以内的素数*/
  B[1] =2； B[2] = 3； B[3] = 5； B[4] = 7；
  /*若A[i]＝i是B[ ]中任一数的倍数，则剔除
  A[i]*/
  for ( j = 1； j <= 4； j++ )
    检查A[ ]所有的数能否被B[j]整除并将
    能被整除的数从A[ ]中剔除； -----3.1
```

1 - 67

```
  /*输出A[ ]中所有没有被剔除的数*/
  for ( i = 2； i <= 100； i++ )
    若A[i]没有被剔除，则输出之 ---4.1
}
```

- 对框架中的局部再做细化，得到整个程序。

1 - 68

```
main ( ) {
  /*建立2到100的数组A[ ]，其中A[i]＝i*/
  for ( i = 2； i <= 100； i++ ) A[i] = i；
  /* 建立2到10的素数表B[ ]，其中存放2
  到10以内的素数*/
  B[1] =2； B[2] = 3； B[3] = 5； B[4] = 7；
  /*若A[i]＝i是B[ ]中任一数的倍数，则剔
  除A[i]*/
  for ( j = 1； j <= 4； j++ )
   /*检查A[ ]所有的数能否被B[j]整除并将
  能被整除的数从A[ ]中剔除*/
```



```
    for ( i = 2； i <= 100； i++）
      if ( A[i] / B[j] * B[j] == A[i] )
          A[i] = 0；
          /*输出A[ ]中所有没有被剔除的
          数*/
   for ( i = 2； i <= 100； i++）
        /*若A[i]没有被剔除，则输出之*/
   if ( A[i] != 0 ）
     printf ( “A[%d]＝%d\n”， i， A[i] )；
  }
```

自顶向下，逐步求精方法的优点

- 符合人们解决复杂问题的普遍规律。可提高软件开发的成功率和生产率
- 用先全局后局部，先整体后细节，先抽象后具体的逐步求精的过程开发出来的程序具有清晰的层次结构，程序容易阅读和理解
- 程序自顶向下，逐步细化，分解成一个树形结构。在同一层的节点上的细化工作相互独立。有利于编码、测试和集成
- 程序清晰和模块化，使得在修改和重新设计一个软件时，可复用的代码量最大
- 每一步工作仅在上层节点的基础上做不多的设计扩展，便于检查
- 有利于设计的分工和组织工作。

课外实践1-1

- 给定的自然数n，确定满足下述关系的最小的数s，s可表示为两对不同的自然数的n次方之和，即找出最小的数s，使得
  - $s=a^n+b^n=c^n+d^n$

i \ j	1	2	3	4	5	6	7	8
1	2							
2	5	8						
3	10	13	18					
4	17	20	25	32				
5	26	29	34	41	50			
6	37	40	45	52	61	72		
7	50	53	58	65	74	85	98	
8	65	68	73	80	89	100	113	128

- 50=12+ 72 =52 +52
- 65=12+ 82 =42 +72
- 50=22+ 92 =62 +72
- ……
- 所求最小数s=50

- 令$s_{ij}=i^n+ j^n$
- 性质1： s_{ij}〉s_{ik},对所有的j及所有的j>k
- 性质2： s_{ij}〉s_{kj},对所有的j及所有的i>k
- 性质3： $s_{ij}=s_{ji}$,对所有的i ,j

N=3

- 第一步，根据所述基本想法，得到程序

```
X=2;
do{
    min=x;
    x="下一个较大的候选者";
}while( x!=min);
write(min);
```

- 第二步，进一步求精

```
for(k=1; k <= ?; k++)
   {
      j[k]=1; s[k]=p(k)+1;
   }
do{
   min=s[i];

1:  “增大j[i]，且用第i行种下一个候选者替换s[i]”
2: “确定i的新值作为具有最小候选者那行的下标”

}while(s[i]!=min);
write(min);
```

- 第三步,对赋初值部分进行简化,得到程序

```
i=1;ih=2;
j[1]=1; s[1]=2; j[2]=1; s[2] =p(2)+1;
do{
   min=s[i];

1:  if j[i]=1 then “增大ih，且给s[ih]置初值”；
2:  “增大j[i]，且用第i行中下一个候选者替换s[i]”
3:  “确定i,使得s[i]=min(s[1],..., s[ih])”

}while(s[i]!=min);
write(min);
```

- 第四步,进一步求精,得到程序

```
i=1;ih=2;i1=1;
j[1]=1; s[1]=2; j[2]=1; s[2] =p(2)+1;
do{
   min=s[i];
   if(j[i]==i) i1=i1+1 else
      {
        1:   if j[i]=1 then “增大ih，且给s[ih]置初值”；
        2:   “增大j[i]，且用第i行中下一个候选者替换s[i]”
      }
        3:        “确定i,使得s[i]=min(s[1],..., s[ih])”

}while(s[i]!=min);
write(min);
```

- 1,2,3可以直接编写出来

```
1: if( j[i]=i )
   {
      ih= ih+1; j[h]=1;s[ih]=p[ih]+1;
   }
 2: j[i]= j[i]+1; s[i]=p[i]+p(j[i]);
 3:i=i1;k=i;
   while(k<ih )do
   {
      k=k+1;
      if( s[k]<s[i] )   i=k;
   };
```

- 第五步,进一步简化
  - 在计算s(i)时,多次用到函数p(i)
  - 避免多余计算,将p(i)保留起来
  - 设置一整型数组p,使得p[i]=p(i)= i * i *i
- 得到完整的程序


```
#include<iostream>
#include<string>
using namespace std;
int main(){
   int i,i1,ih,min,a,b,k;
   int j[13],s[13],p[13];
   i = 1; ih = 2; i1 = 1;
   j[1] = 1; p[1] = 1; s[1] = 1;
   j[2] = 1; p[2] = 8; s[2] = 9;
   do{
      min = s[i]; a = i; b = j[i];
      if(j[i] == i)
         i1 = i1 +1;
      else{
         if(j[i] == 1){
            ih = ih + 1;p[ih] = ih * ih * ih;
            j[ih] = 1;s[ih] = p[ih] + 1;
         }
```

```
            j[i] = j[i] + 1; s[i] = p[i] + p[j[i]];
        }
        i = i1; k = i;
        while(k < ih){
            k = k + 1;
            if(s[k] < s[i])
                i = k;
        }
    }
    while(s[i] != min);
    cout << "s = "<< min << " = " << a << "^3 + " << b << "^3 = " << i
<< "^3 + " << j[i] << "^3" << endl;
    return 0;
}
```

课外实践1-2

- (八皇后问题)

课外实践1-3

- p.55 2-19 某人到商店买一件衣服，其价为19元，可此人只有2元钞票若干张，而售货员只有5元钞票若干张。问这笔钱怎么付（即整张钞票找齐）？给出这问题的程序。
- p.55 2-21 验证歌德巴赫猜想：所有大于2的偶数，均可表示成两个素数之和。
  - 程序验证到N=100，并按4=2+2
  - 6=3+3
  - 8=3+5
  - …
  - 格式打印输出。

主程序员组的组织形式...

- 组成
  - 主程序员：1人
    - 整个软件系统的开发，是该组的技术经理。主程序员将亲自完成软件系统的关键部分的设计、编码和调试
  - 辅助程序员：1人
    - 在细节上给主程序员充分支持，可取代主程序员
    - 要有程序设计技术、项目管理经验
  - 程序资料员（或秘书）：1人
  - 外加的程序员、系统分析员和技术员将依据项目的规模和类型，据需要随时加入

...主程序员组的组织形式

- 好处
  - 减少各子系统间的各程序模块之间的通讯和接口方面的问题

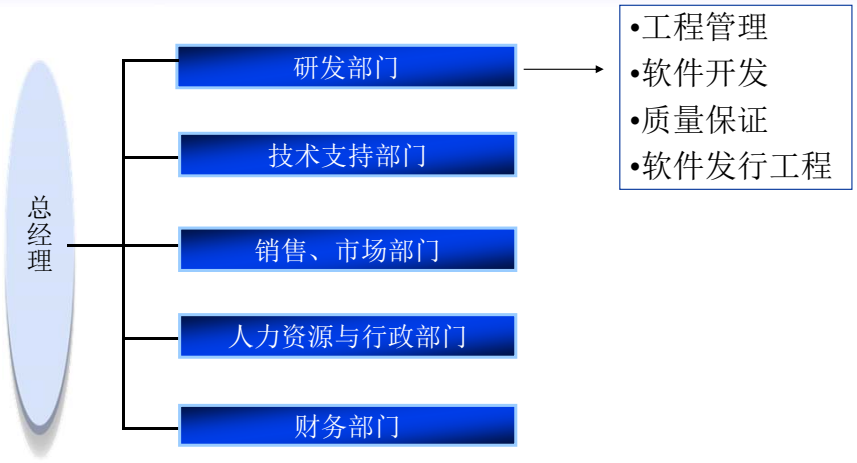
- HOARE

将人们的思维按这样一种方式组织起来，使之在合理的时间内，将一个计算任务表示成容易理解的形式。这样一种工作就是“结构程序设计”。

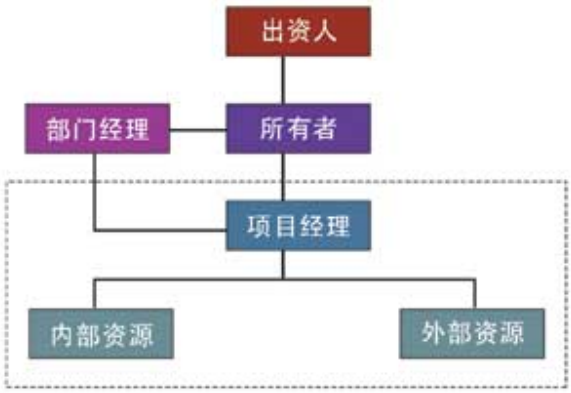
是团队，而不是个人，创造了软件。

- 世界上有四个人：
- **每个人**、**某个人**、**任何人**和**没有人**。
- 有一项重要的任务，
- 要求**每个人**去做，
- **每个人**都以为**某个人**会去做。
- 本来**任何人**都可以完成这项工作，
- 但是**没有人**去做。
- **某个人**对此感到愤怒，
- 因为这是**每个人**的工作。
- **每个人**以为**任何人**都可以完成这项工作，
- 但是**没有人**意识到**每个人**不可能完成它。
- 最后**每个人**都责备**某个人**，
- 因为**没有人**做**任何人**都可以做到的事。

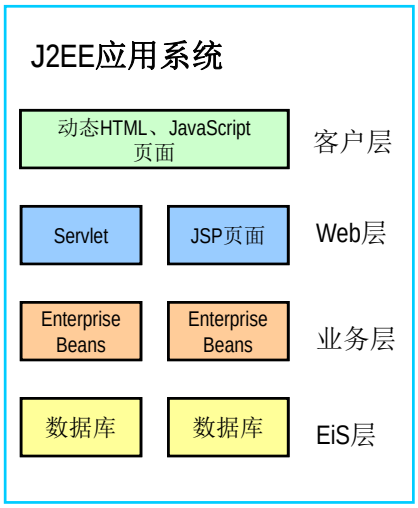
软件公司组织结构



组建项目团队



组建J2EE项目开发团队



- 设计师
- 表现逻辑层
  - Java Servlet开发人员
  - JSP开发人员
  - HTML设计人员
  - 美工人员
  - 客户端的JavaScript开发人员
- 业务逻辑层
  - Session Bean开发人员
  - Entity Bean开发人员
- 数据库访问层
- 设计用户界面
- EiS：企业信息系统

开发过程团队沟通：Rational ClearQuest …

- ClearQuest：针对软件开发的动态性和交互性而设计的项目管理工具
- 作用
  - 加强开发团队内部的沟通
  - 管理和软件开发有关的任何类型的活动在实际项目管理中
- 用途：
  - BaseCMActivity：表示基本配置管理活动
  - EnhancementRequest：用于涉众请求或新需求
  - Defect：表示缺陷

…开发过程团队沟通：Rational ClearQuest

- 使用 Rational 工具进行项目开发、设计和管理的最佳模式
  - 需求人员使用ClearQuest收集项目需求→汇总成为需求文档→纳入RequisitePro管理→项目经理根据需求确定计划→通过ClearQuest生成任务工单→项目开发人员利用 ClearQuest 或 ClearCase 激活的任务→如果建模则使用Rose→完成后将工作产品提交到ClearCase→测试人员使用Rational测试工具套件完成测试任务→测试人员利用ClearQuest记录缺陷→ClearCase 自动记录配置项的修改和变更→项目管理人员分别使用ClearQuest生成缺陷和任务工单状态报告，使用RequisitePro生成需求跟踪报告，使用ClearCase生成配置管理报告