武汉大学
WUHAN UNIVERSITY

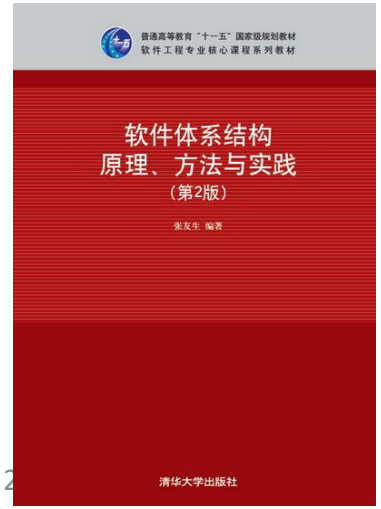
第1章 软件体系结构概论

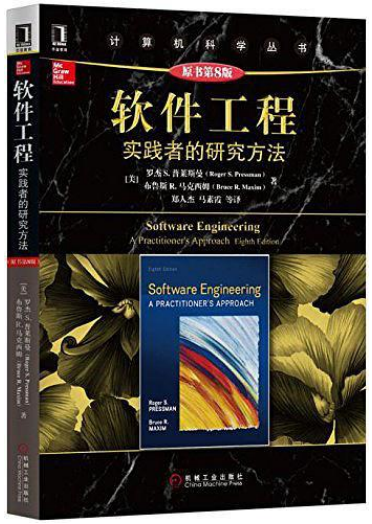
授课教师： 应时

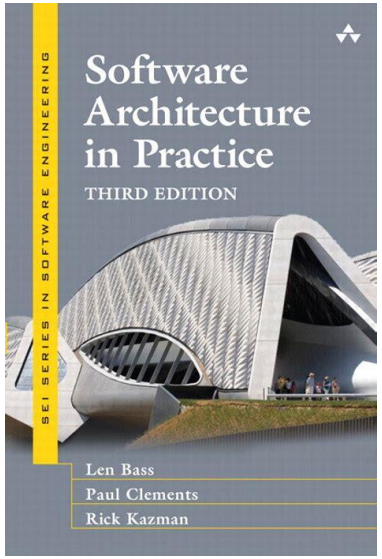
2019-10-15

参考书

- 张友生编著，软件体系结构原理、方法与实践（第2版），清华大学出版社，2014年1月。
- [美]罗杰 S. 普莱斯曼，译者: 郑人杰，软件工程：实践者的研究方法（原书第8版），机械工业出版社，2016年11月。
- 温昱，软件架构设计（第2版）—程序员向架构师转型必备，电子工业出版社，2012年07月。
- Len Bass, Paul Clements, Rick Kazman, Software Architecture in Practice (3rd Edition), Addison-Wesley Professional, Sep 25, 2012.





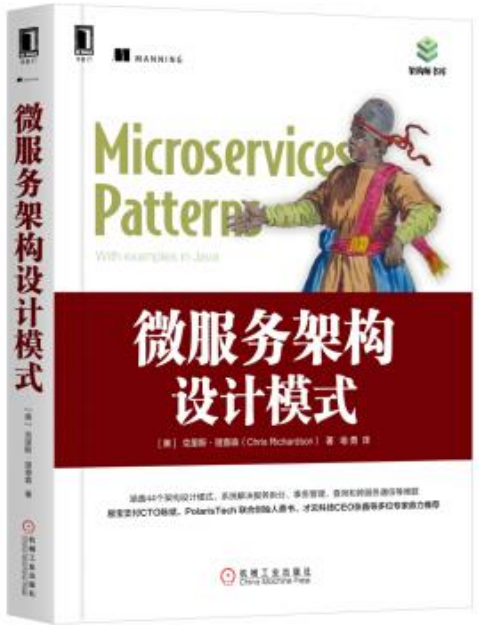


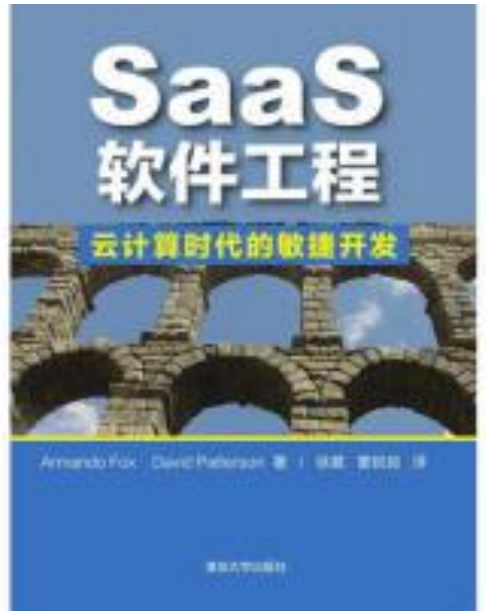
参考书

- 软件建模与设计：UML、用例、模式和软件体系结构，原书名：Software Modeling & Design: UML, Use Cases, Patterns, & Software Architectures，作者：(美)Hassan Gomaa，译者： 彭鑫、吴毅坚、赵文耘，机械工业出版社，2014 年8月。
- 软件架构建模和仿真：Palladio 方法，原书名：Modeling and Simulating Software Architectures: The Palladio Approach，作者： （德）拉尔夫•H.雷乌斯纳（Ralf H.Reussner） 等，译者： 李必信 等，机械工业出版社，2018 年9月。

参考书

- Chris Richardson著，喻勇译，微服务架构设计模式，机械工业出版社，2019年5月。
- Armando Fox、David Patterson 著，徐葳、曹锐创译，SaaS软件工程：云计算时代的敏捷开发，清华大学出版社，2015年6月。





参考书

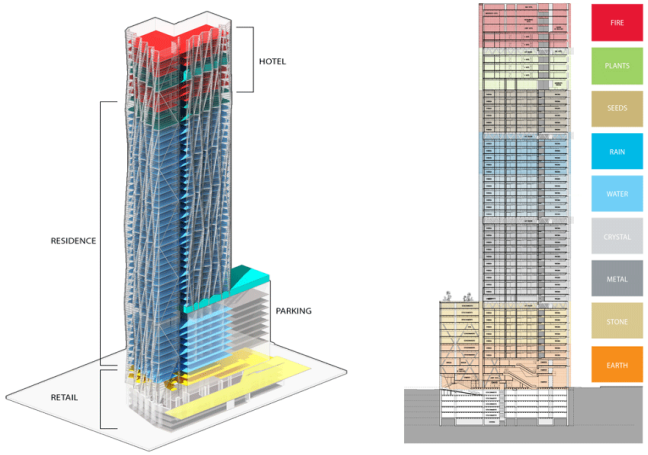
- Edward Crawley等 著，爱飞翔译，系统架构--复杂系统的产品设计与开发，机械工业出版社，2017年1月。

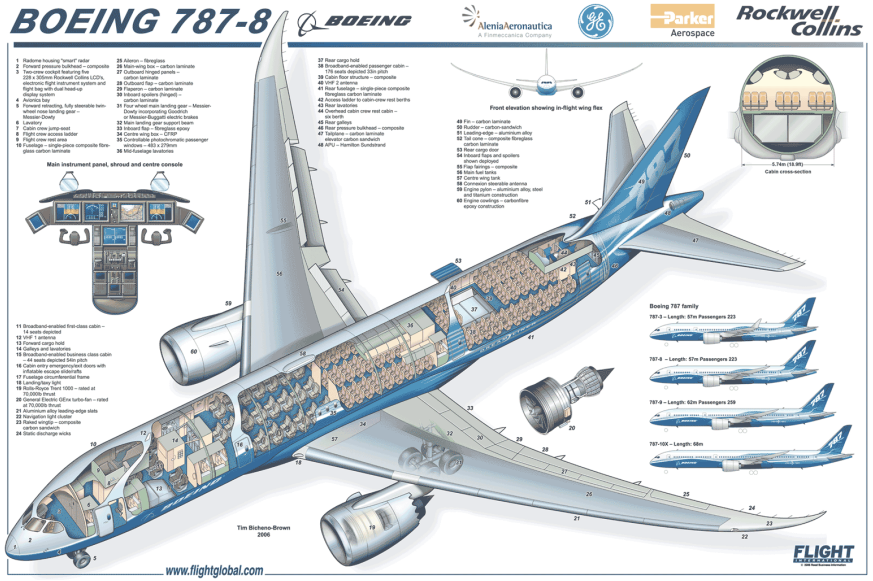
课程学习目标

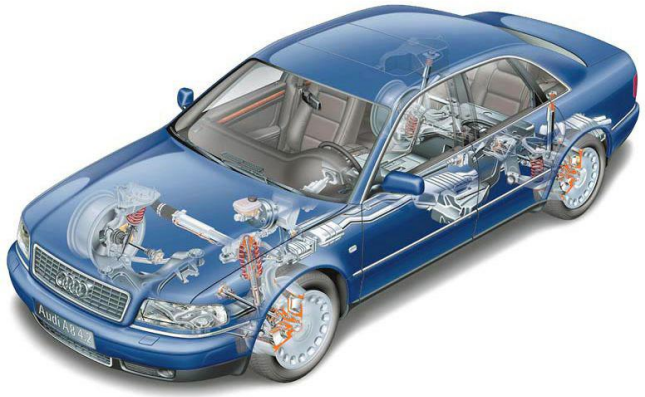
- **技能应用型：精通技术知识，能做实用性开发工作**
  - 掌握软件体系结构设计的知识，熟悉软件体系结构的风格和模式等知识。
  - 熟悉软件体系结构的4+1视图模型，能够熟练地使用UML及其支撑工具（例如，IBM Rational），完成实际项目开发中的软件体系结构的设计、描述、演化等工作任务。
  - 能够对软件体系结构设计方案，进行权衡分析和质量评价。

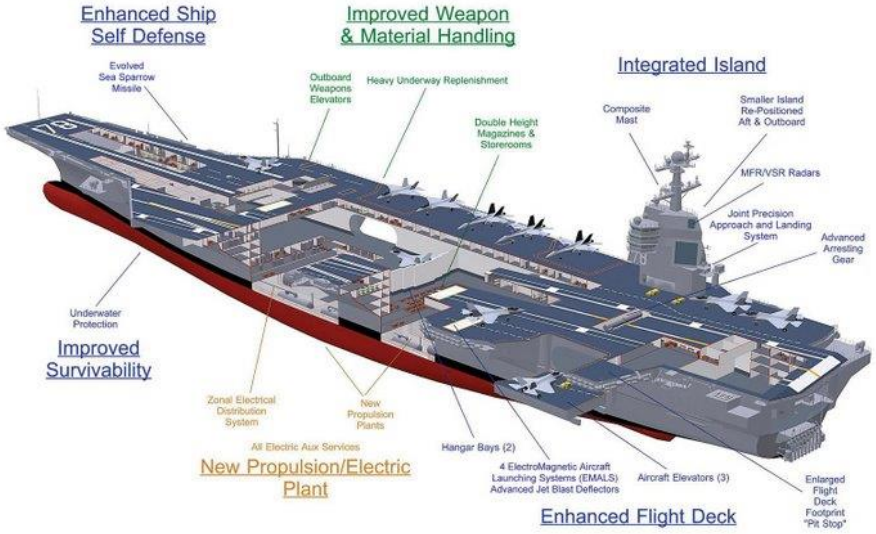
- **基础研究型：掌握理论知识，能做基础性研究工作**
  - 掌握体系结构描述语言ADL知识，熟悉一二种ADL及其形式化理论基础，能熟练地应用ADL及其支撑工具，对软件体系结构进行形式化地描述、分析和验证。
  - 掌握软件体系结构保障软件质量的机理与方法。

体系结构



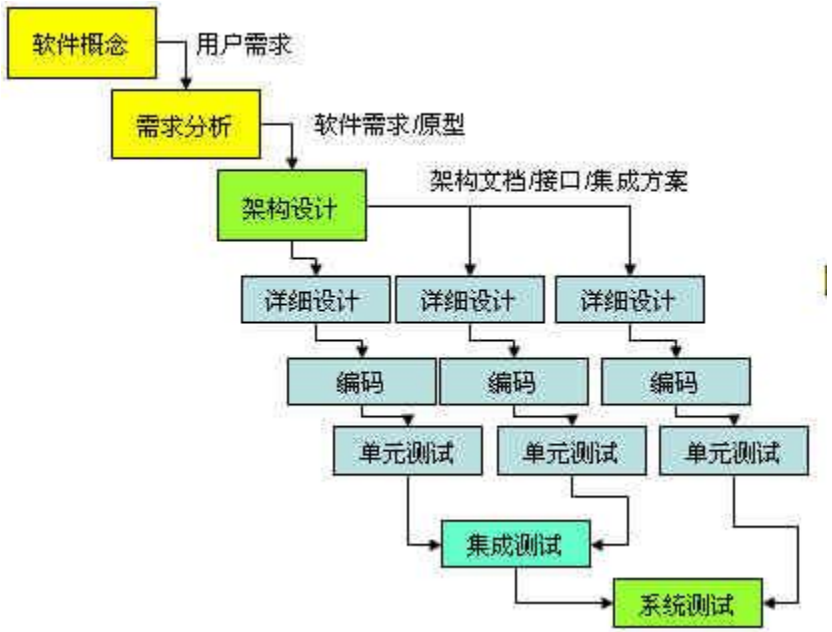






注：本页部分图引自《覃征，软件体系结构课件》

软件体系结构设计



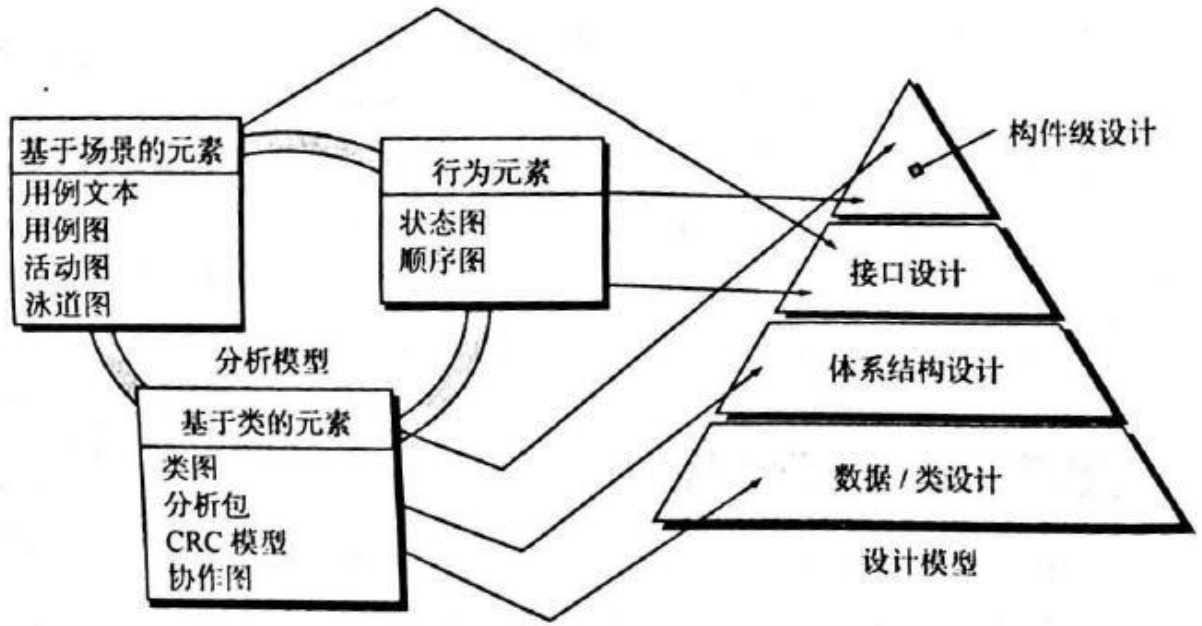
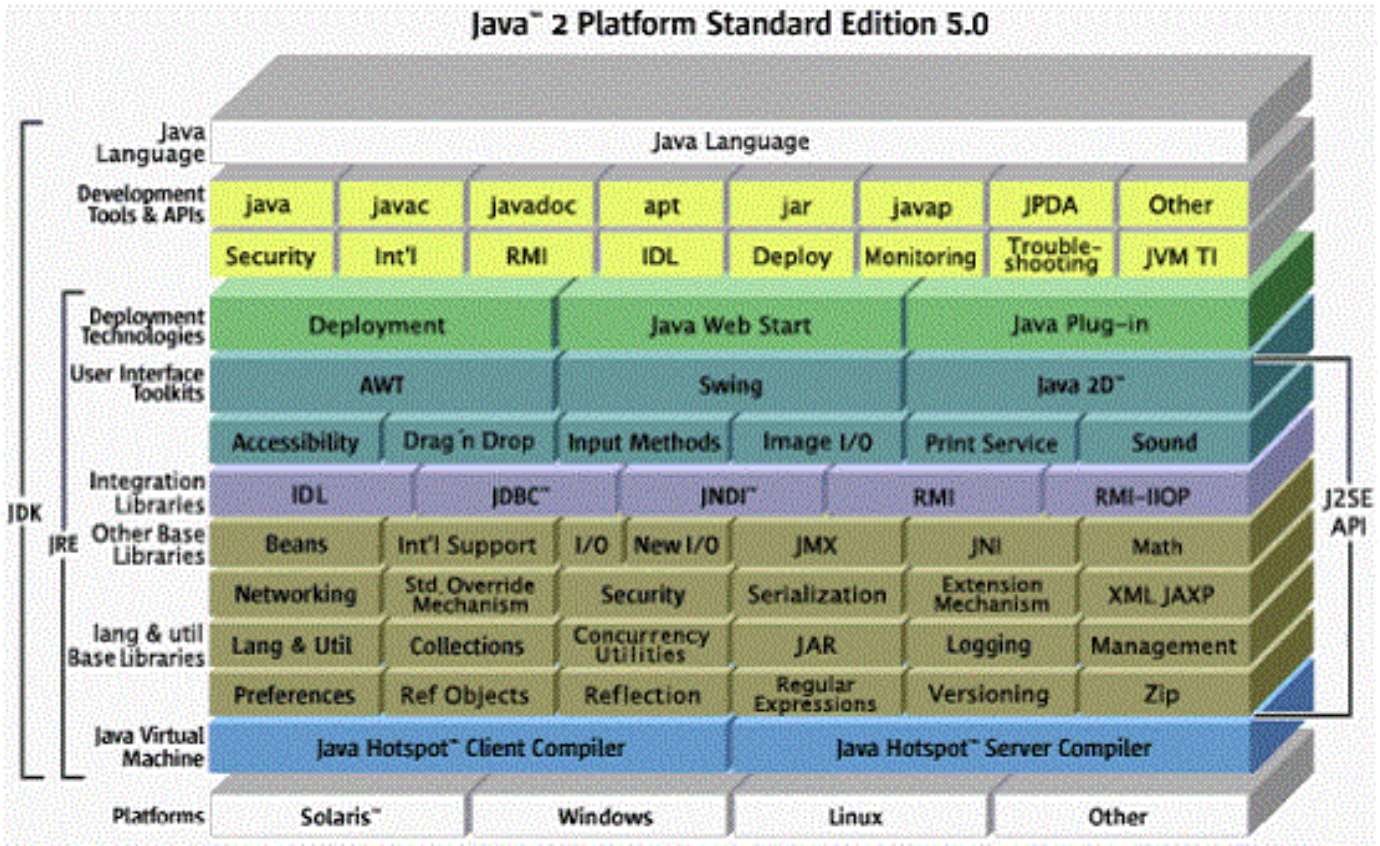
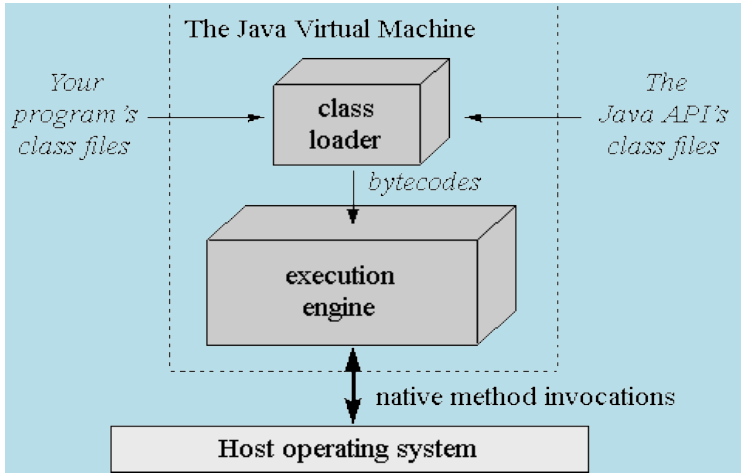


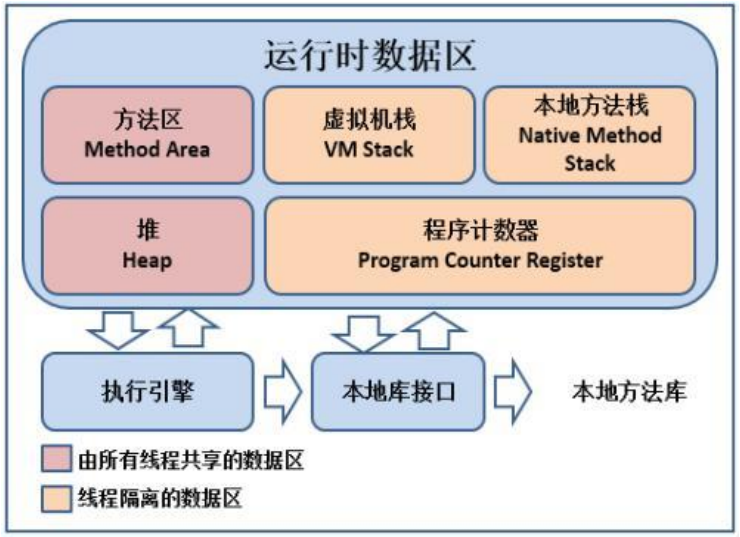
图 12-1 从需求模型到设计模型的转换

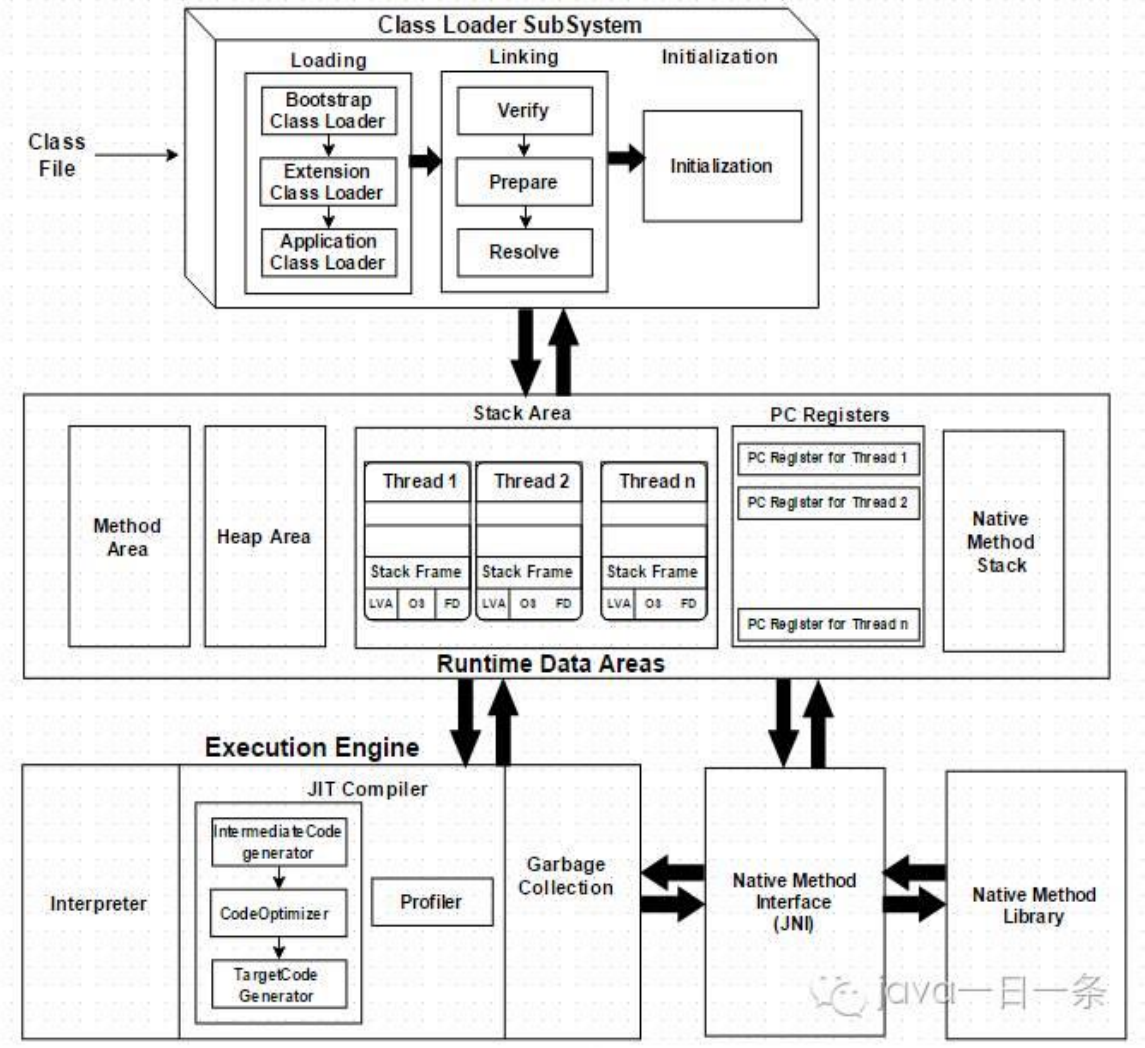
Java平台的体系结构



JVM的体系结构

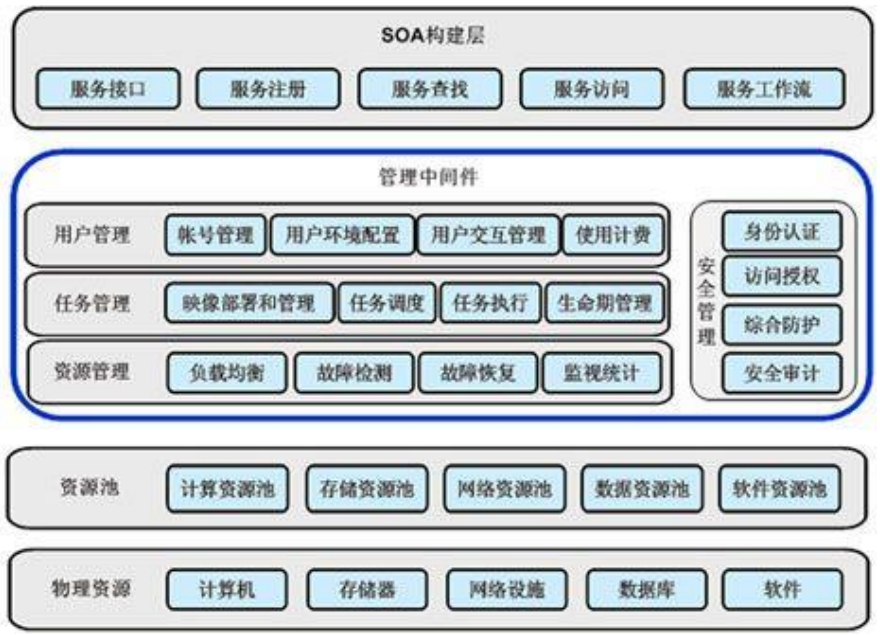

The Java Virtual Machine
Your program's class files
class loader
The Java API's class files
bytecodes
execution engine
native method invocations
Host operating system

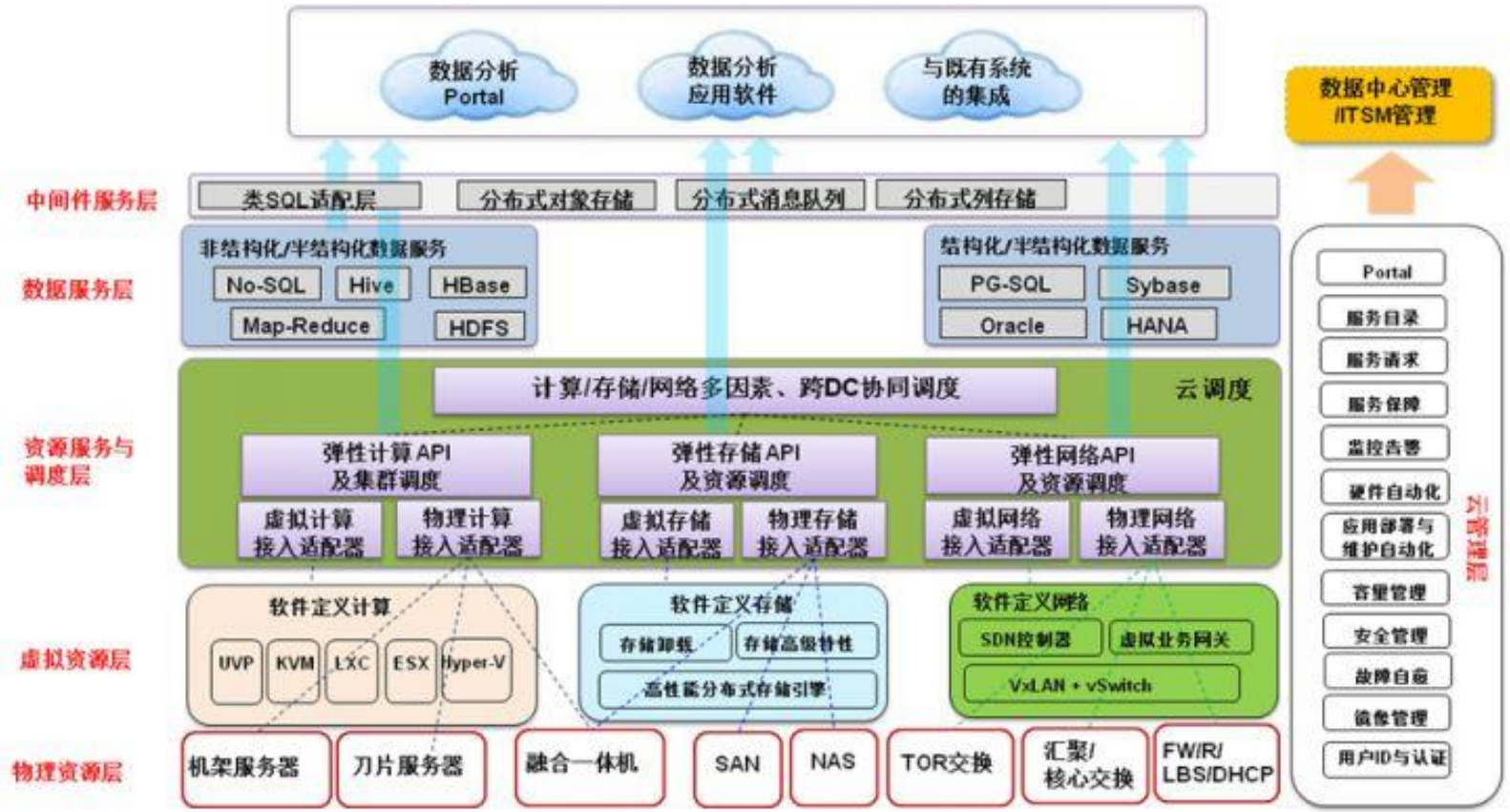

运行时数据区
方法区
Method Area
虚拟机栈
VM Stack
本地方法栈
Native Method Stack
堆
Heap
程序计数器
Program Counter Register
执行引擎
本地库接口
本地方法库
由所有线程共享的数据区
线程隔离的数据区


Class Loader SubSystem
Loading
Linking
Initialization
Class File
Bootstrap Class Loader
Extension Class Loader
Application Class Loader
Verify
Prepare
Resolve
Initialization
Stack Area
PC Registers
Method Area
Heap Area
Thread 1
Thread 2
Thread n
Stack Frame
LVA
OS
FD
PC Register for Thread 1
PC Register for Thread 2
PC Register for Thread n
Native Method Stack
Runtime Data Areas
Execution Engine
Interpreter
JIT Compiler
Intermediate Code generator
CodeOptimizer
TargetCode Generator
Profiler
Garbage Collection
Native Method Interface (JNI)
Native Method Library
java一日一条

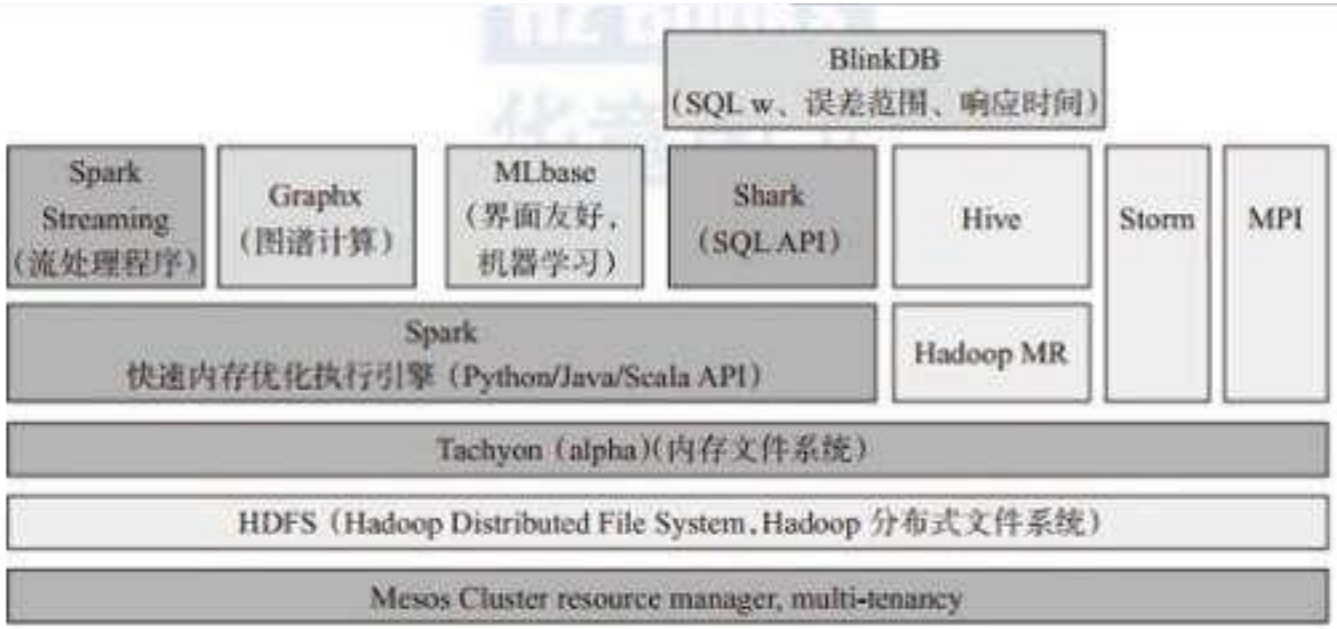
云计算体系结构

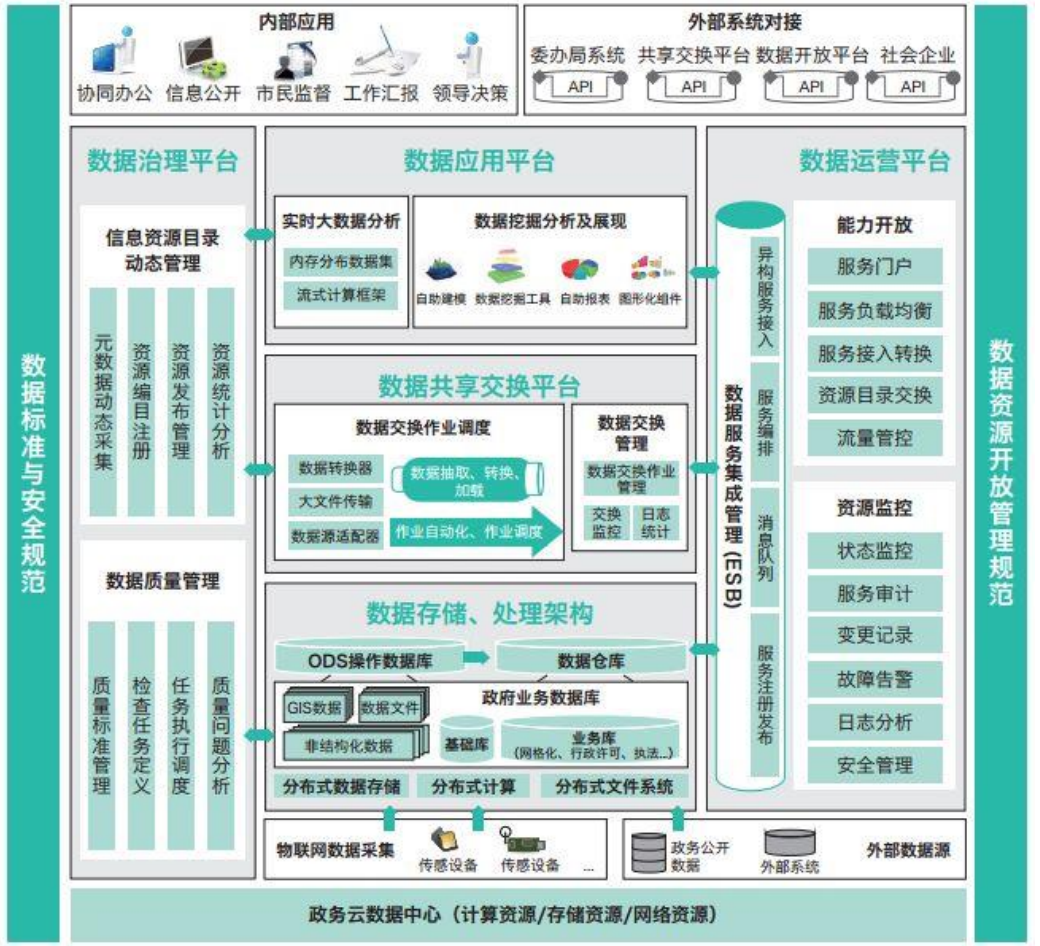
图 2-1 云计算的架构





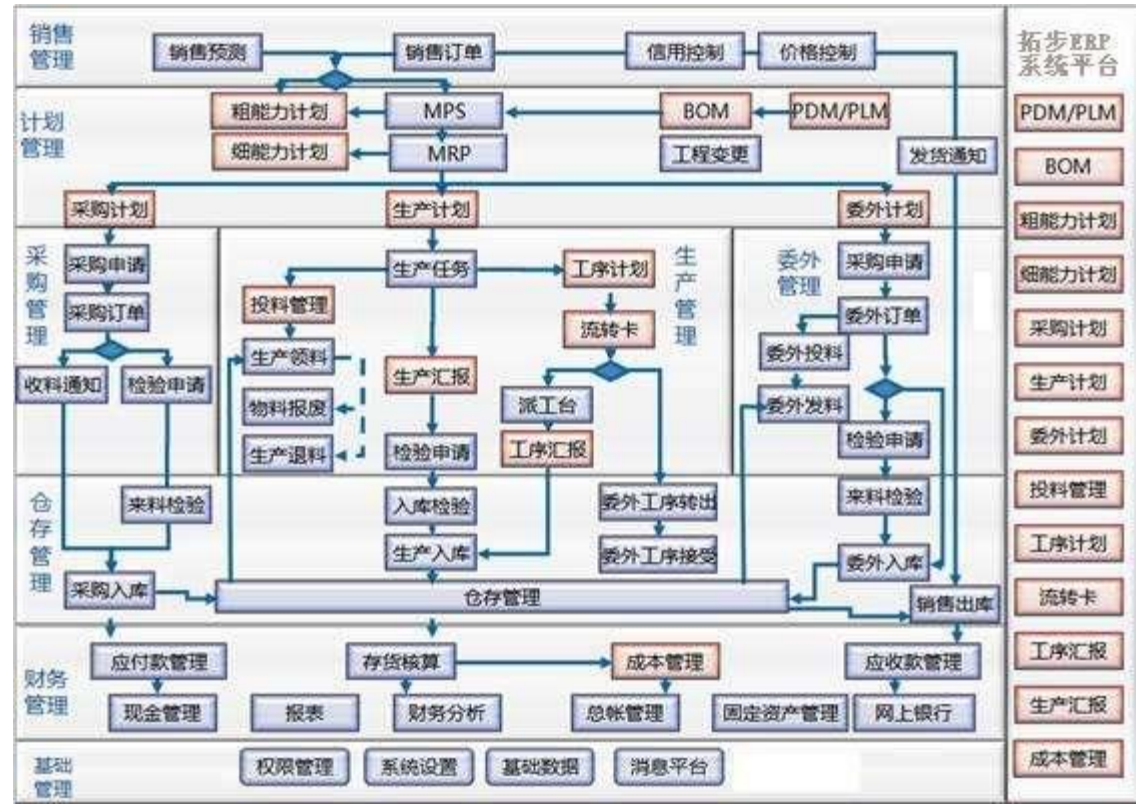
Spark与数据服务体系结构





ERP软件体系结构


协同办公
文件管理
通知公告
流程审批
短信设置
短信设置
报表自动分发
密码与订阅
采购管理
物料需求管理
采购申请管理
供应商报价管理
采购订单管理
材料交期变更
采购追踪与预警
物料管理
入库暂收管理
正式入库管理
生产领料管理
生产补料管理
综合出/入库管理
库存盘点管理
委外管理
材料委外
工艺委外
订单委外
品质管理
材料检验
车间检验
成品返修
财务管理
存货核算管理
应付款管理
应收款管理
成本管理
财务接口管理
人力资源
考勤管理
人事管理
计件管理
工资管理
人权管理
生产管理
生产产能管理
檀模设定和管理
排期推导设定
材料参数设定
订单总进度管理
生产指令管理
生产周计划管理
生产日报管理
裁断日安排
面部日安排
成型日安排
生产各类报表
现场管理
生产派工管理
中转仓管理
现场工票管理
派工领料管理
条码/RFID应用
派工日报管理
开发管理
开发前期管理
样品制作管理
开发物料管理
样品报价管理
开发费用管理
开发BOM管理
试做管理
量产BOM管理
新型体试做进度
量产试做单
试做材料管理
试做计划管理
试做报告管理
订单与发货管理
订单导入管理
正式订单管理
正式报价管理
成品入库管理
订单发货管理
成品装箱管理
EOS平台
工作流设计
消息引擎
规则引擎
对象设计
功能设计
界面设计
集成引擎


销售管理
销售预测
销售订单
信用控制
价格控制
计划管理
粗能力计划
细能力计划
MPS
MRP
BOM
PDM/PLM
工程变更
发货通知
采购计划
生产计划
委外计划
采购管理
采购申请
采购订单
收料通知
检验申请
生产任务
投料管理
生产领料
物料报废
生产退料
生产汇报
检验申请
工序计划
流转卡
派工台
工序汇报
生产管理
委外管理
采购申请
委外订单
委外投料
委外发料
检验申请
来料检验
委外入库
仓存管理
来料检验
采购入库
入库检验
生产入库
仓存管理
委外工序转出
委外工序接受
销售出库
财务管理
应付款管理
现金管理
报表
存货核算
财务分析
成本管理
总帐管理
固定资产管理
应收款管理
网上银行
基础管理
权限管理
系统设置
基础数据
消息平台
拓步ERP系统平台
PDM/PLM
BOM
粗能力计划
细能力计划
采购计划
生产计划
委外计划
投料管理
工序计划
流转卡
工序汇报
生产汇报
成本管理

- 软件危机
- 软件危机的表现
- 1、软件成本日益增长
  - 20世纪50年代，软件成本在整个计算机系统成本中所占的比例为10%-20%。但随着软件产业的发展，软件成本日益增长。
  - 相反，计算机硬件随着技术的进步、生产规模的扩大，价格却在不断下降。这样一来，软件成本在计算机系统中所占的比例越来越大。

- 到20世纪60年代中期，软件成本在计算机系统中所占的比例已经增长到50%左右。而且，该数字还在不断地递增。
- 下面是一组来自美国空军计算机系统的数据：
  - 1955年，软件费用约占总费用的18%，
  - 1970年达到60%，
  - 1975年达到72%，
  - 1980年达到80%，
  - 1985年达到85%左右。
- 而如今，购买一台普通的电脑只要2-3千元人民币，但如果把常用的操作系统、办公软件、安全软件等装好，却要远远超过购买电脑的费用。

1.1.1 软件危机的表现

➢ 2、开发进度难以控制

- 在软件开发过程中，用户需求变化等各种意想不到的情况层出不穷，令软件开发过程很难保证按预定的计划实现，给项目计划和论证工作带来了很大的困难。

1.1.1 软件危机的表现

- Brooks在其名著《人月神话》中指出：“在已拖延的软件项目上，增加人力只会使其更难按期完成”。
  - 事实上，软件系统的结构很复杂，各部分附加联系极多，盲目增加软件开发人员并不能成比例地提高软件开发能力。相反，随着人员数量的增加，人员的组织、协调、通信、培训和管理等方面的问题将更为严重。
- 许多重要的大型软件开发项目，如IBM OS/360和世界范围的军事命令控制系统（WWMCCS），在耗费了大量的人力和财力之后，由于离预定目标相差甚远，不得不宣布失败。

1.1.1 软件危机的表现

- 3、软件质量差
- 软件项目即使能按预定日期完成，结果却不尽人意。
  - 1965年至1970年，美国范登堡基地发射火箭多次失败，绝大部分故障是由应用程序错误造成的。程序的一些微小错误可以造成灾难性的后果，
  - 例如，有一次，在美国肯尼迪发射一枚阿脱拉斯火箭，火箭飞离地面几十英里高空开始翻转，地面控制中心被迫下令炸毁。后经检查发现是飞行计划程序里漏掉了一个连字符。就是这样一个小小的疏漏造成了这支价值1850万美元的火箭试验失败。
- 在“软件作坊”里，由于缺乏工程化思想的指导，程序员几乎总是习惯性地以自己的想法，去代替用户对软件的需求，软件设计带有随意性，很多功能只是程序员的“一厢情愿”而已，这是造成软件不能令人满意的重要因素。

1.1.1 软件危机的表现

- 4、软件维护困难
- 缺标准、缺文档：
  - 由于在软件设计和开发过程中，没有严格遵循软件开发标准，各种随意性很大，没有完整的真实反映系统状况的记录文档，给软件维护造成了巨大的困难。
- 开发人员流失：
  - 特别是在软件使用过程中，原来的开发人员可能因各种原因，已经离开原来的开发组织，使得软件几乎不可维护。
- 软件修改是一项很“危险”的工作：
  - 对一个复杂的逻辑过程，哪怕做一项微小的改动，都可能引入潜在的错误，常常会发生“纠正一个错误带来更多新错误”的问题，从而产生副作用。
- 有资料表明，工业界为维护软件支付的费用占全部硬件和软件费用的40%-75%。

1.1.2 软件危机的原因

- 1、用户需求不明确
- 在软件开发过程中，需求多变是造成项目失败的最主要原因之一。
- 具体来说，用户需求不明确问题主要体现在四个方面：

① 在软件开发出来之前，用户自己也不清楚软件的具体需求。

② 用户对软件需求的描述不精确，可能有遗漏、有二义性、甚至有错误。

③ 在软件开发过程中，用户还提出修改软件功能、界面、支撑环境等方面的要求。

④ 软件开发人员对用户需求的理解与用户本来愿望有差异。

1.1.2 软件危机的原因

- 2、缺乏正确的、全面的理论指导
  - 缺乏有力的方法学和工具方面的支持。
  - 软件开发过程是复杂的逻辑思维过程，其产品极大程度地依赖于开发人员高度的智力投入。
  - 过分地依靠开发人员在软件开发过程中的技巧和创造性，加剧软件产品的个性化。

1.1.2 软件危机的原因

- 3、软件规模越来越大
- 随着软件应用范围的增广，软件规模愈来愈大。
  - 大型软件项目需要组织一定的人力共同完成，而多数管理人员缺乏开发大型软件系统的经验，而多数软件开发人员又缺乏管理方面的经验。
  - 各类人员的信息交流不及时、不准确、有时还会产生误解。
  - 软件项目开发人员不能有效地、独立自主地处理大型软件的全部关系和各个分支，因此容易产生疏漏和错误。

1.1.2 软件危机的原因

- 4、软件复杂度越来越高
- 软件不仅仅是在规模上快速地发展扩大，而且其复杂性也急剧地增加。
- 软件产品的特殊性和人类智力的局限性，导致人们无力处理“复杂问题”。
- 所谓“复杂问题”的概念是相对的，一旦人们采用先进的组织形式、开发方法和工具提高了软件开发效率和能力，新的、更大的、更复杂的问题又摆在人们的面前。

1.1.3 如何克服软件危机

- 人们在认真地研究和分析了软件危机背后的真正原因之后，得出了“人们面临的不单是技术问题，更重要的还是管理问题。管理不善必然导致失败”的结论，便开始探索用工程的方法进行软件生产的可能性，即用现代工程的概念、原理、技术和方法进行计算机软件的开发、管理和维护。
- 于是，计算机科学技术的一个新领域——软件工程（software engineering）诞生了。

1.1.3 如何克服软件危机

- 软件工程是：用工程、科学和数学的原则与方法，研制和维护计算机软件的有关技术及管理方法。
- 软件工程包括三个要素：方法、工具和过程，其中：
  - 软件工程方法为软件开发提供了“如何做”的技术，是完成软件工程项目的技术手段。
  - 软件工具是人们在开发软件的活动中智力和体力的扩展和延伸，为软件工程方法提供了自动的或半自动的软件支撑环境。
  - 软件工程过程则是将软件工程的方法和工具综合起来以达到合理、及时地进行计算机软件开发的目的。
- 迄今为止，软件工程的研究与应用已经取得很大成就。它在软件开发方法、工具、管理等方面的应用大大缓解了软件危机造成的被动局面。

1.2 构件与软件重用

- 从80年代中后期开始，人们认识到，要提高软件开发效率，提高软件产品质量，必须采用工程化的开发方法与工业化的生产技术。
- 这包括技术与管理两方面的问题：
  - 在技术上，应该采用基于重用的软件生产技术
  - 在管理上，应该采用多维的工程管理模式

1.2 构件与软件重用

- 分析传统工业及计算机硬件产业成功的模式可以发现，这些工业的发展模式均是符合标准的零部件/构件生产，以及基于标准构件的产品生产。其中，构件是核心和基础，重用是必需的手段。
- 实践表明，这种模式是产业工程化、工业化的成功之路，也将是软件产业发展的一条有前景的道路。

1.2 构件与软件重用

- 软件重用是指在两次或多次不同的软件开发过程中，重复使用相同或相近软件元素的过程。
- 软件元素包括程序代码、测试用例、设计文档、设计过程、需求分析文档甚至领域（domain）知识。
- 通常，把这种可重用的元素称作软构件（software component），简称为构件。
- 可重用的软件元素越大，就说重用的粒度（granularity）越大。

1.2 构件与软件重用

- 使用软件重用技术可以减少软件开发活动中大量的重复性工作，这样就能提高软件生产率，降低开发成本，缩短开发周期。
- 同时，由于软构件大都经过严格的质量认证，并在实际运行环境中得到检验，因此，重用软构件有助于改善软件质量。此外，大量使用软构件，软件的灵活性和标准化程度也能得到提高。

1.2.1 构件模型及实现

- 一般认为，构件是指语义完整、语法正确和有可重用价值的单位软件，是软件重用过程中可以明确辨识的系统；结构上，它是语义描述、通讯接口和实现代码的复合体。
- 简单地说，构件是具有一定的功能，能够独立工作，或能同其它构件装配起来协调工作的程序体，构件的使用同它的开发、生产无关。
- 从抽象程度来看，面向对象（Object Orientation，OO）技术已达到了类级重用（代码重用），它以类为封装的单位。这样的重用粒度还太小，不足以解决异构互操作和效率更高的重用。
- 构件将抽象的程度提到一个更高的层次，它是对一组类的组合进行封装，并代表完成一个或多个功能的特定服务，也为用户提供了多个接口。整个构件隐藏了具体的实现，只用接口对外提供服务。

1.2.1 构件模型及实现

- 构件模型（model）是对构件本质特征的抽象描述。
- 目前，国际上已经形成了许多构件模型，这些模型的目标和作用各不相同，其中部分模型属于参考模型（例如，3C模型），部分模型属于描述模型（例如，RESOLVE模型和REBOOT模型）。还有一部分模型属于实现模型。
- 近年来，已形成三个主要流派，分别是OMG（Object Management Group，对象管理组织）的CORBA（Common Object Request Broker Architecture，通用对象请求代理结构）、Sun的EJB（Enterprise Java Bean）和Microsoft的DCOM（Distributed Component Object Model，分布式构件对象模型）。
- 这些实现模型将构件的接口与实现进行了有效的分离，提供了构件交互（interaction）的能力，从而增加了重用的机会，并适应了目前网络环境下大型软件系统的需要。

1.2.1 构件模型及实现

- 国内许多学者在构件模型的研究方面做了不少的工作，取得了一定的成绩，其中较为突出的是北京大学杨芙清院士等人提出的“青鸟构件模型”。
- 青鸟构件模型充分吸收了上述模型的优点，并与它们相容。青鸟构件模型由外部接口（interface）与内部结构两部分组成，如图1-1所示。

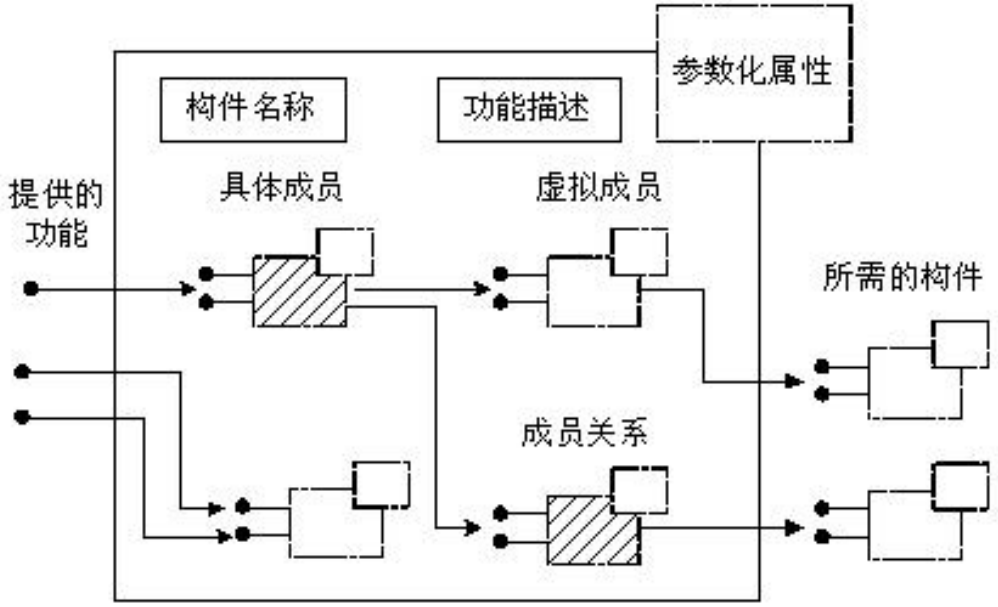


图1-1 青鸟构件模型

1.2.1 构件模型及实现

- 1、外部接口
- 构件的外部接口是指构件向其重用者提供的基本信息，包括：构件名称、功能描述、对外功能接口、所需的构件、参数化属性等。
- 外部接口是构件与外部世界的一组交互点，说明了构件所提供的那些服务（消息、操作、变量）。

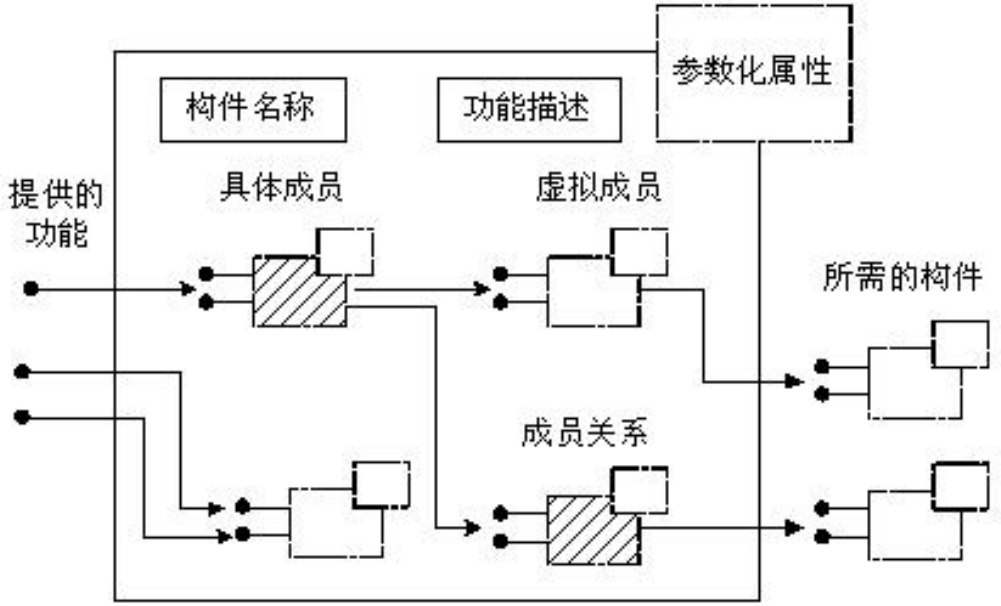


图1-1 青鸟构件模型

1.2.1 构件模型及实现

- 2、内部结构
- 构件的内部结构包括两方面内容：内部成员以及内部成员之间的关系。其中内部成员包括具体成员与虚拟成员，而成员关系包括内部成员之间的互联，以及内部成员与外部接口之间的互联。
- 构件实现是指具体实现构件功能的逻辑系统，通常也称为代码构件。构件实现由构件生产者完成，构件重用者则不必关心构件的实现细节。重用者在重用构件时，可以对其定制，也可以对其特例化。

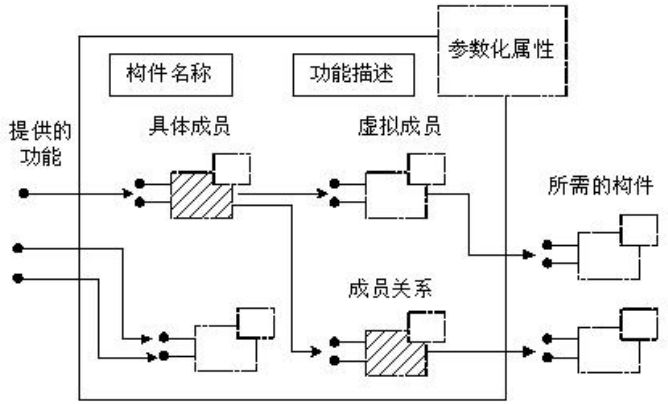


图1-1 青鸟构件模型

1.2.1 构件模型及实现

- **构件的3C模型**
- **概念(Concept)– 关于“构件做什么”的抽象描述**
  - 通过概念，去理解构件的功能。
  - 概念包括：接口规约和语义描述两个部分
- **内容(Content)– 概念的具体实现**
  - 描述构件如何完成和实现概念所刻划的功能。
- **语境(Context)– 构件和外围环境，在概念级和内容级的关系**
  - 语境刻划构件的应用环境。
  - 为构件的选用和适应性修改，提供指导。

1.2.2 构件获取

- 存在大量的可重用的构件是有效地使用重用技术的前提。
- 通过对可重用信息与领域的分析，可以得到：
  - 可重用信息具有领域特定性，即可重用性不是信息的一种孤立的属性，它依赖于特定的问题和特定的问题解决方法。为此，在识别（identify）、获取（capture）和表示（represent）可重用信息时，应采用面向领域的策略。
  - 领域具有内聚性和稳定性，即关于领域的解决方法是充分内聚和充分稳定的。一个领域的规约和实现知识的内聚性，使得可以通过一组有限的、相对较少的可重用信息来解决大量问题。
  - 领域的稳定性，使得获取的信息可以在较长的时间内多次重用。

1.2.2 构件获取

- 领域是一组具有相似或相近软件需求的应用系统所覆盖的功能区域
- 领域工程（domain engineering）是基于一组相似或相近系统的应用工程（application engineering）的成果，而建立基本能力和必备基础的过程。
- 领域工程过程可划分为：
  - 领域分析
  - 领域设计
  - 领域实现等多个活动
  - 其中的活动与结果如图1-2所示。

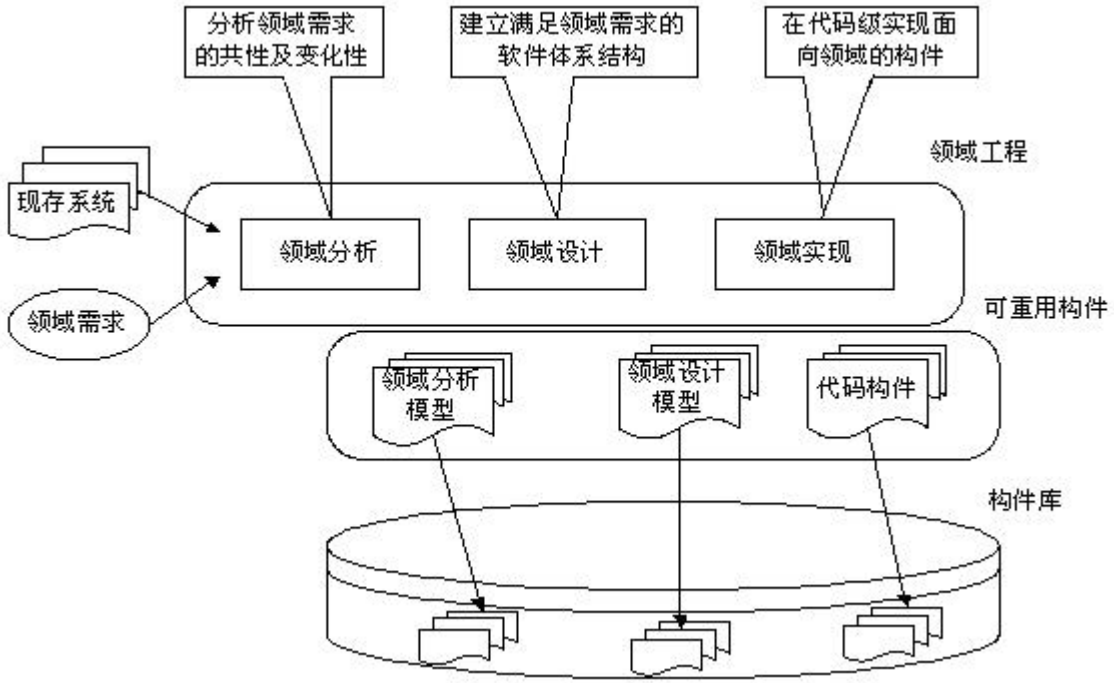


图 1-2 领域工程中的活动与结果

1.2.2 构件获取

- 在建立基于构件的软件开发（Component-Based Software Development，CBSD）中，**构件获取可以有多种不同的途径**：
  1. 从现有构件中，获得符合要求的构件，直接使用或作适应性（flexibility）修改，得到可重用的构件。
  2. 通过遗留工程（legacy engineering），将具有潜在重用价值的构件提取出来，得到可重用的构件。
  3. 从市场上购买现成的商业构件，即COTS（Commercial Off-The-Shell）构件。
  4. 开发新的符合要求的构件。
- 一个组织在进行以上决策时，必须考虑到不同方式获取构件的一次性成本和以后的维护成本，然后做出最优的选择。

1.2.3 构件管理

- 对大量的构件进行有效的管理，以方便构件的存储、检索和提取，是成功重用构件的必要保证。
- 构件管理的内容包括：
  - 构件描述
  - 构件分类
  - 构件库组织
  - 人员及权限管理
  - 用户意见反馈等

1.2.3 构件管理

- 1、构件描述
- 构件模型是对构件本质的抽象描述，主要是为构件的制作与构件的重用提供依据。
- 从管理角度出发，也需要对构件进行描述。
  - 例如：实现方式、实现体、注释、生产者、生产日期、大小、价格、版本和关联构件等信息。它们与构件模型一起，共同组成了对构件的完整描述。

1.2.3 构件管理

➢ 2、构件分类与组织

- 为了给使用者在查询构件时提供方便，同时也为了更好地重用构件，就必须对收集和开发的构件进行分类（classify），并置于构件库的适当位置。
- 构件的分类方法及相应的库结构，对构件的检索和理解有极为深刻的影响。
- 因此，构件库的组织应方便构件的存储和检索。

1.2.3 构件管理

- 可重用技术对构件库组织方法的要求是：

1. 支持构件库的各种维护动作，如增加、删除以及修改构件，尽量不要影响构件库的结构。
2. 不仅要支持精确匹配，还要支持相似构件的查找。
3. 不仅能进行简单的语法匹配，而且能够查找在功能或行为方面，等价或相似的构件。
4. 对应用领域具有较强的描述能力和较好的描述精度。
5. 库管理员和用户容易使用。

1.2.3 构件管理

- 目前，已有的构件分类方法可以归纳为三大类，分别是：
  - 关键字分类法
  - 刻面分类法
  - 超文本组织方法

1.2.3 构件管理

- （1）关键字分类法
- 关键字分类法是一种最简单的构件库组织方法。其基本思想是：根据领域分析的结果，将应用领域的概念，按照从抽象到具体的顺序，逐次分解为树形，或有向无回路图结构。
- 每个概念用一个描述性的关键字表示。
- 概念一直可以被分解到不可分解的原子级。
- 图1-3给出了构件库的关键字分类结构示例，它支持图形用户界面设计。

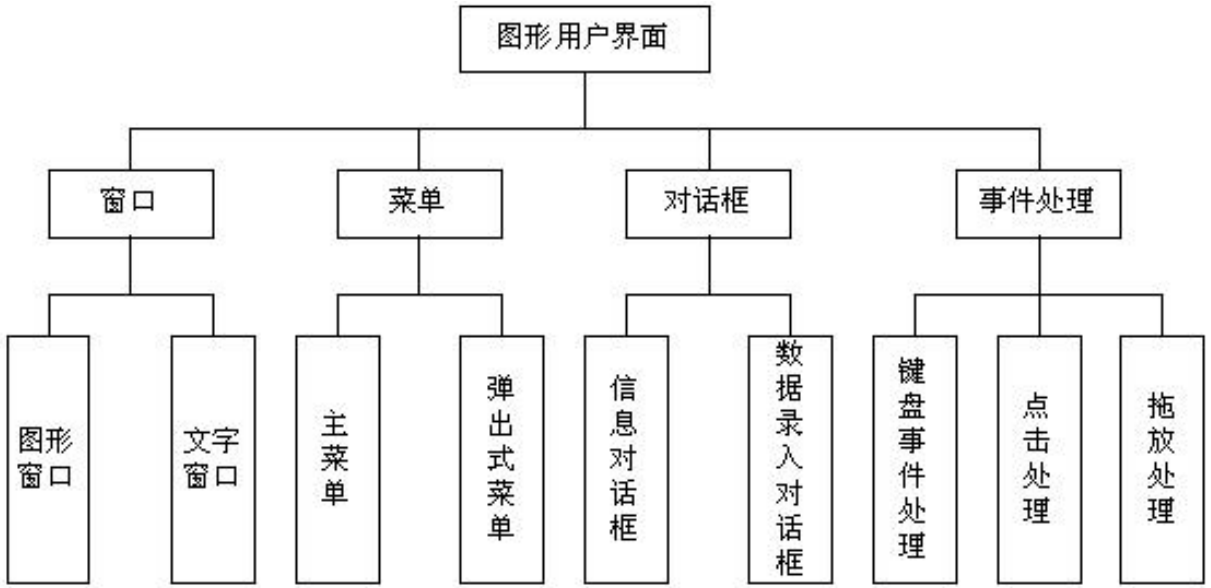


图1-3 关键字分类结构示例

1.2.3 构件管理

- 当加入构件时，库管理员必须对构件的功能或行为进行分析，在浏览上述关键字分类结构的同时，将构件置于最合适的原子级关键字之下。
- 如果无法找到构件的属主关键字，可以扩充现有的关键字分类结构，引进新的关键字。
- 但库管理员必须保证，新关键字有相同的领域分析结果，作为支持。
  - 例如，如果需要增加一个“图形文字混合窗口”构件时，则只需把该构件放到属主关键字“窗口”的下一级。

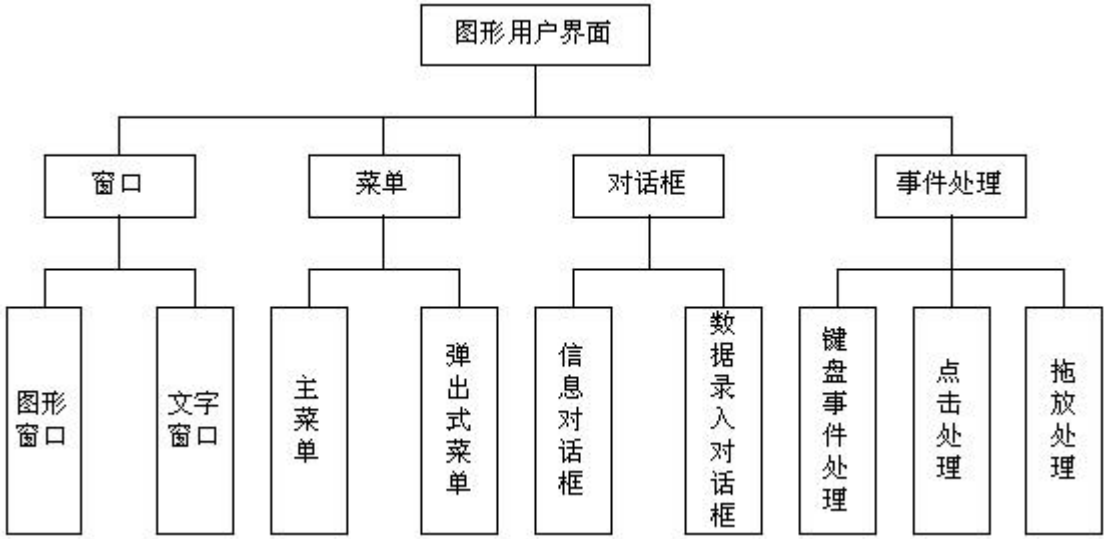


图1-3 关键字分类结构示例

1.2.3 构件管理

- （2）刻面分类法
- 刻面分类法（faceted classification）的主要思想来源于图书馆学，这种分类方法是Prieto-Diaz和Freeman在1987年提出来的。
- 在刻面分类机制中，定义若干用于刻画构件特征的“面”（facet），每个面包含若干概念，这些概念表述构件在面上的特征。（注：“面”对应于概念）
- 刻面可以描述构件执行的功能、被操作的数据、构件应用的语境或任意其他特征。
- 描述构件的刻面的集合称为刻面描述符（facet descriptor），通常，刻面描述被限定不超过7或8个刻面。当描述符中出现空的特征值时，表示该构件没有相应的面。

1.2.3 构件管理

- 作为一个简单的在构件分类中使用刻面的例子，考虑使用下列构件描述符的模式：

  {function，object type，system type}

- 刻面描述符中的每个刻面可含有一个或多个值，这些值一般是描述性关键词，
  - 例如，如果功能是某构件的刻面，赋给此刻面的典型值可能是：function = (copy，from) or (copy，replace，all)
- 多个刻面值的使用，使得原函数copy，能够被更完全地进行细化描述。

1.2.3 构件管理

- 关键词(值)被赋给重用库中的每个构件的刻面集，当软件工程师在设计中希望查询构件库，以发现可能的构件时，规定一列值，然后到库中寻找匹配项。
- 可使用自动工具，以完成同义词词典功能，这使得查找不仅包括软件工程师给出的关键词，还包括这些关键词的技术同义词。

1.2.3 构件管理

- 青鸟构件库就是采用刻面分类方法对构件进行分类的，这些刻画包括：

1. 使用环境。使用（包括理解/组装/修改）该构件时，必须提供的硬件和软件平台。
2. 应用领域。构件原来或可能被使用到的应用领域（及其子领域）的名称。
3. 功能。在原有或可能的软件系统中，所提供的软件功能集合。
4. 层次。构件相对于软件开发过程阶段的抽象层次，如分析、设计、编码等。
5. 表示方法。用来描述构件内容的语言形式或媒体，如源代码构件所用的编程语言环境等。

1.2.3 构件管理

- 关键字分类法和刻面分类法都是以数据库系统作为实现背景。
- 尽管关系数据库可供选用，但面向对象数据库更适于实现构件库，因为其中的复合对象、多重继承等机制，与表格相比，更适合描述构件及其相互关系。

1.2.3 构件管理

- **分面组配式分类法—来源于图书分类学**
- **分面组配式分类法是根据概念的分析和综合原理，编制的文献分类法，又称分面分类法、组配分类法、分析--综合分类法。**
- **它将主题概念分解为简单概念(或概念因素)，按照它们所属的方面或范畴，分别编列成表。**
- **标引时，用两个或多个简单概念的分类号的组合，来表达一个复杂的主题概念。**

1.2.3 构件管理

- （3）超文本组织方法
- 超文本方法（hypertext classification）与基于数据库系统的构件库组织方法不同，它基于全文检索（full text search）技术。
- 其主要思想是：
  - 所有构件必须辅以详尽的功能或行为说明文档；
  - 说明中出现的重要概念或构件，以网状链接方式，相互连接；
  - 检索者在阅读文档的过程中，可按照人类的联想思维方式，任意跳转到包含相关概念或构件的文档；
  - 全文检索系统，将用户给出的关键字与说明文档中的文字进行匹配，实现构件的浏览式检索。

1.2.3 构件管理

- 超文本是一种非线性的网状信息组织方法，它以结点为基本单位，链作为结点之间的联想式关联，如图1-4所示。

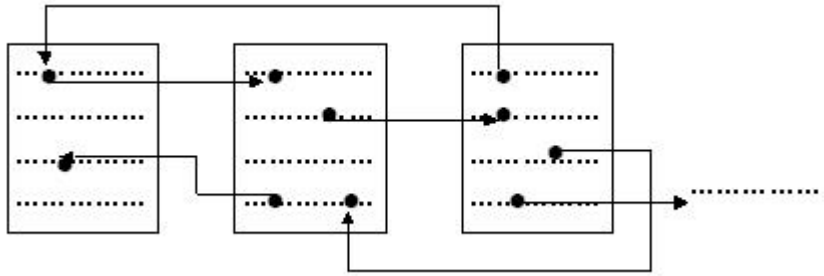


图1-4 超文本结构示意图

- 一般地，结点是一个信息块。
- 对可重用构件而言，结点可以是域概念、功能或行为名称、构件名称等。
- 在图形用户界面上，结点可以是字符串，也可以是图象、声音和动画等。
- 超文本组织方法，为构造构件和重用构件，提供了友好、直观的多媒体方式。
- 由于网状结构比较自由、松散，因此，超文本方法比前两种方法，更易于修改构件库的结构。

1.2.3 构件管理

- 例如，Windows环境下的联机帮助系统就是一种典型的超文本系统。
- 为构造构件的文档，
  - 首先，要根据领域分析的结果，在说明文档中，标识超文本结点并在相关文档中建立链接关系；
  - 然后，用类似于联机帮助系统编译器的工具，对构件的说明文档，进行编译；
  - 最后，用相应的工具（例如：IE浏览器）运行编译后的目标即可。

1.2.3 构件管理

- 如果把软件系统看成是构件的集合，那么从构件的外部形态来看，**构成一个系统的构件可分为5类：**

1. **独立而成熟的构件。**独立而成熟的构件得到了实际运行环境的多次检验。该类构件隐藏了所有接口，用户只需用规定好的命令进行使用。例如，数据库管理系统和操作系统等。
2. **有限制的构件。**有限制的构件提供了接口，指出了使用的条件和前提，这种构件在装配时，会产生资源冲突、覆盖等影响，在使用时需要加以测试。例如，各种面向对象程序设计语言中的基础类库等。
3. **适应性构件。**适应性构件进行了包装或使用了接口技术，对不兼容性、资源冲突等进行了处理，可以直接使用。这种构件可以不加修改地使用在各种环境中。例如ActiveX等。

1.2.3 构件管理

- 如果把软件系统看成是构件的集合，那么从构件的外部形态来看，构成一个系统的构件可分为5类：

4. 装配的构件。装配（assemble）的构件在安装时，已经装配在操作系统、数据库管理系统或信息系统不同层次上，使用胶水代码（glue code）就可以进行连接使用。目前一些软件商提供的大多数软件产品都属这一类。
5. 可修改的构件。可修改的构件可以进行版本替换。如果对原构件修改错误、增加新功能，可以利用重新“包装”或写接口来实现构件的替换。这种构件在应用系统开发中使用得比较多。

1.2.3 构件管理

- 3、人员及权限管理
- 构件库系统是一个开放的公共构件共享机制，任何使用者都可以通过网络访问构件库，这在为使用者带来便利的同时，也给系统的安全性带来了一定的风险。
- 因此有必要对不同使用者的访问权限（privilege）作出适当的限制，以保证数据安全。

1.2.3 构件管理

- 构件库系统可包括5类用户
  - 注册用户
  - 公共用户
  - 构件提交者
  - 一般系统管理员
  - 超级系统管理员

1.2.3 构件管理

- 他们对构件库分别有不同的职责和权限，这些人员相互协作，共同维护着构件库系统的正常运作。
- 同时，系统为每一种操作定义一个权限，包括提交构件、管理构件、查询构件及下载构件。
- 每一用户可被赋予一项或多项操作权限，这些操作权限组合，形成该人员的权限，从而支持对操作的分工，为权限分配提供了灵活性。

1.2.4 构件重用

- 构件开发的目的是重用，为了让构件在新的软件项目中发挥作用，库的使用者必须完成以下工作：
  - 检索与提取构件
  - 理解与评价构件
  - 修改构件
  - 最后，将构件组装到新的软件产品中

1.2.4 构件重用

- 1、检索与提取构件
- 构件库的检索方法与组织方式密切相关，下面针对关键字分类法、刻面分类法和超文本组织方法，分别讨论相应的检索方法。
- （1）基于关键字的检索
  - 这种简单检索方法的基本思想是：系统在图形用户界面上，将构件库的关键字树形结构直观地展示给用户；用户通过对树形结构的逐级浏览，寻找需要的关键字并提取相应的构件。当然，用户也可直接给出关键字（其中可含通配符），由系统自动给出合适的候选构件清单。
  - 这种方法的优点是简单、易于实现。但在某些场合没有应用价值，因为用户往往无法用构件库中已有的关键字，描述期望的构件功能或行为，对库的浏览也容易使用户迷失方向。

1.2.4 构件重用

- （2）刻面检索法
- 该方法基于刻面分类法，由三步构成：
  - 第一步：构造查询。用户提供要查找的构件在每个刻面上的特征，生成构件描述符。此时，用户可以从构件库已有的概念中进行挑选，也可将某些特征值指定为空。
  - 第二步：检索构件。实现刻面检索法的计算机辅助软件工程工具，在构件库中，寻找相同或相近的构件描述符及相应的构件。
  - 第三步：对构件进行排序。被检索出来的构件清单，除按相似程度排序外，还可以按照与重用有关的度量信息排序。
    - ✓ 例如，构件的复杂性、可重用性、已成功的重用次数等。
  - 这种方法的优点是它易于实现相似构件的查找，但用户在构造查询时比较麻烦。

1.2.4 构件重用

- （3）超文本检索法
- 超文本检索法的基本步骤是：用户首先给出一个或数个关键字，系统在构件的说明文档中进行精确或模糊的语法匹配，匹配成功后，向用户列出相应的构件说明。
- 构件说明是含有许多超文本结点的正文，用户阅读这些正文时，可实现多个构件说明文档之间的自由跳转，最终选择合适的构件。
  - 为了避免用户在跳转过程中迷失方向，系统可以通过图形界面，提供浏览历史信息图，允许将特定画面，定义为命名“书签”，并随时跳转至“书签”，并帮助用户逆跳转路径，而逐步返回。
- 这种方法的优点是用户界面友好，但在某些情况下，用户难以在超文本浏览过程中，正确选取构件。

1.2.4 构件重用

- （4）其他检索方法
- 上述检索方法基于语法（syntax）匹配，要求使用者对构件库中出现的众多词汇，有较全面的把握、较精确的理解。
- 理想的检索方法是语义（semantic）匹配：构件库的用户以形式化（formalization）手段，描述所需要的构件的功能或行为语义；系统通过定理证明及基于知识的推理过程，寻找语义上等价或相近的构件。
- 遗憾的是，这种基于语义的检索方法涉及许多人工智能难题，目前尚难于支持大型构件库的工程实现。

1.2.4 构件重用

- 2、理解与评价构件
- 要使库中的构件在当前的开发项目中发挥作用，准确地理解构件是至关重要的。
- 当开发人员需要对构件进行某些修改时，情况更是如此。
- 考虑到设计信息对于理解构件的必要性，以及构件的用户逆向发掘设计信息的困难性，必须要求构件的开发过程，遵循公共软件工程规范，并且在构件库的文档中，全面、准确地说明以下内容：

1. 构件的功能与行为
2. 相关的领域知识
3. 可适应性约束条件与例外情形
4. 可以预见的修改部分及修改方法

1.2.4 构件重用

- 但是，如果软件开发人员希望重用以前并非专为重用而设计的构件，上述假设即不能成立。
- 此时开发人员必须借助于CASE工具，对候选构件进行分析。
  - 这种CASE工具对构件进行扫描，将各类信息存入某种浏览数据库，然后回答构件用户的各类查询，进而帮助理解。

- 逆向工程是理解构件的另一种重要手段。
- 它试图通过对构件的分析，结合领域知识，半自动地生成相应的设计信息，然后借助设计信息，完成对构件的理解和修改。

1.2.4 构件重用

- 对构件可重用的评价，是通过收集并分析构件的用户在实际重用该构件的历史过程中的各种反馈信息来完成的。
- 这些信息包括：
  - 重用成功的次数
  - 对构件的修改量
  - 构件的健壮性度量
  - 性能度量等。

1.2.4 构件重用

➢ 3、修改构件

- 理想的情形是对库中的构件不作修改，而直接用于新的软件项目。
- 但是，在大多数情况下，必须对构件进行或多或少的修改，以适应新的需求。

1.2.4 构件重用

- 3、修改构件
- 为了减少构件修改的工作量，要求开发人员尽量使构件的功能、行为和接口设计更为抽象化、通用化和参数化。这样，构件的用户即可通过对实参的选取，来调整构件的功能或行为。
- 如果这种调整仍不足以使构件适用于新的软件项目，用户就必须借助设计信息和文档，来理解、修改构件。所以，与构件有关的文档和抽象层次更高的设计信息对于构件的修改至关重要。
  - 例如，如果需要将C语言书写的构件，改写为Java语言形式，构件的算法描述就十分重要。

1.2.4 构件重用

- 4、构件组装
- 构件组装是指将库中的构件，经适当修改后相互连接，或者将它们与当前开发项目中的软件元素相连接，最终构成新的目标软件。
- 构件组装技术大致可分为：
  - 基于功能的组装技术
  - 基于数据的组装技术
  - 面向对象的组装技术

1.2.4 构件重用

- （1）基于功能的组装技术
  - 基于功能的组装技术采用子程序调用和参数传递的方式，将构件组装起来。
  - 它要求库中的构件以子程序/过程/函数的形式出现，并且接口说明必须清晰。
  - 当使用这种组装技术进行软件开发时，开发人员首先应对目标软件系统进行功能分解，将系统分解为强内聚、松耦合的功能模块。
  - 然后根据各模块的功能需求提取构件，对它进行适应性修改后，再挂接在上述功能分解框架（framework）中。

1.2.4 构件重用

- （2）基于数据的组装技术
  - 基于数据的组装技术，首先根据当前软件问题的核心数据结构，设计出一个框架，然后根据框架中各结点的需求，提取构件，并进行适应性修改，再将构件逐个分配至框架中的适当位置。
  - 此后，构件的组装方式，仍然是传统的子程序调用与参数传递。
  - 这种组装技术也要求库中构件以子程序形式出现，但它所依赖的软件设计方法不再是功能分解，而是面向数据的设计方法，例如，Jackson系统开发方法。

1.2.4 构件重用

- （3）面向对象的组装技术
  - 由于封装和继承特征，面向对象方法很适合支持软件重用。
  - 在面向对象的软件开发方法中，如果从类库中检索出来的基类，能够完全满足新软件项目的需求，则可以直接应用。
  - 否则，就把类库中的基类作为父类，并采用构造法或子类法，去构造子类。

1.2.4 构件重用

- **方法一：构造法**
- **为了在子类中，使用库中基类的属性和方法，可以考虑在子类中，引进基类的对象，作为子类的成员变量。**
- **然后，在子类中，通过成员变量，重用基类的属性和与方法。**

```
//定义基类
class Person{
public:
    Person(char *name, int age);
    ~Person();
protected:
    char *name;
    int age;
};
```

```
//基类构造函数
Person::Person(char * name, int age)
{
    Person::name = new char[strlen(name) + 1];
    strcpy(Person::name,name);
    Person::age = age;
    cout << “Construct Person” << name << “,” << age << “.\n”;
    return;
}

//基类析构函数
Person:: ~Person()
{
    cout << “Destruct Person” << name << “,” << age << “.\n”;
    delete name;
    return;
}
```

1.2.4 构件重用

- 方法一：构造法

```
//下面采用构造法生成Teacher类
class Teacher{
public:
    Teacher(char *name, int age, char *teaching);
    ~Teacher();
protected:
    Tperson * Person;
    char *course;
};

// Teacher类的实现
Teacher::Teacher(char *name, int age, char *teaching)
{
    //重用基类的方法Person()
    Tperson = new Person(name, age);
    strcpy(course, teaching);
    return;
}
Teacher::~Teacher()
{
    delete Tperson;
}
```

1.2.4 构件重用

- **方法二：子类法**
- **与构造法完全不同，子类法将新子类，直接说明为库中基类的子类。**
- **通过继承和修改基类的属性与行为，完成新子类的定义。**

```
//定义基类
class Person{
public:
    Person(char *name, int age);
    ~Person();
protected:
    char *name;
    int age;
};
```

```
//基类构造函数
Person::Person(char * name, int age)
{
    Person::name = new
char[strlen(name) + 1];
    strcpy(Person::name,name);
    Person::age = age;
    cout << “Construct Person” <<
name << “,” << age << “.\n”;
    return;
}

//基类析构函数
Person:: ~Person()
{
  cout << “Destruct Person” << name
<< “,” << age << “.\n”;
    delete name;
    return;
}
```

1.2.4 构件重用

- 方法二：子类法

```
//下面采用子类法构造Teacher类
class Teacher : public Person{
public:
    Teacher(char *name, int age, char *teaching):Person(name,age)
    {
course = new char[strlen(teaching) + 1];
strcpy(course, teaching);
return;
    }
    ~Teacher();
    {
delete course;
return;
    }
protected:
    char *course;
};
```

1.2.5 构件重用实例

- 针对电子政务领域产品的开发过程中的重用策略与方法，从系统分析、设计到编码，介绍软件的领域重用与层次重用的方法。

1.2.5 构件重用实例

➢ 1、应用背景

- 开发适应新时代的基于Internet和Intranet的电子政务系统。
- 通用办公管理系统（General Office Management System，GOMS）是面向政府机关的信息化管理和办公自动化系统，是电子政务的主要组成部分。
- GOMS和其他电子政务系统一样，存在跨平台、分布、异构以及对原有应用系统进行整合的问题。
- 为了面对各类机关的应用需要，GOMS采用了多层B/S体系结构和J2EE技术实现了系统的分布异构及跨平台，支持流行操作系统、Web服务器以及数据库管理系统。

1.2.5 构件重用实例

➢ 2、需求重用

- 在产品领域定位的指导下，经过深入的分析调研，设计师发现所有的事务型系统都有一个共同的特征：**工作流程**。
  - ✓ 在ISO9000中也规定任何组织的事务处理必须有标准、规范的工作流程。
- **在系统分析时，可以将这些业务工作流程抽象出来**，如公文管理中的收文流程、发文流程、归档流程、稽催流程、档案管理流程等。
- 而且，在非公文管理的其他业务中也可以抽取流程，如：车辆管理业务流程、会议室管理流程、请假加班流程等。
- 因此，**可以建立一个工作流平台，使所有的业务流程只要在工作流平台中进行定义就可以运作。**

1.2.5 构件重用实例

- 一般的应用软件产品除了完成业务所需要的功能外，还必须有一些支持模块，以支持系统的正常运行。
- 这些模块通常包括组织管理模块和系统支持模块。
  - 组织管理是机关业务得以正常运作的基础，这对于每一个电子政务领域内的应用系统来说，都是必不可少的。
  - 通常系统支持模块是为了软件系统的正常运作所提供的必不可少的功能，如系统权限管理、日志管理、数据库备份/恢复功能等都属于此类。
- 对于软件企业来说，保持系列产品在风格上的一致性是非常重要的。它可以降低系列产品的维护、培训费用，而且还可以在软件开发时进行界面风格重用，降低软件开发成本。

1.2.5 构件重用实例

➢ 3、设计重用

- 在产品开发之初，识别了所有的业务流程，都可以运行于工作流平台之上。
- 在设计时，采用了以工作流平台为核心的领域软件产品设计框架，如图1-5所示。

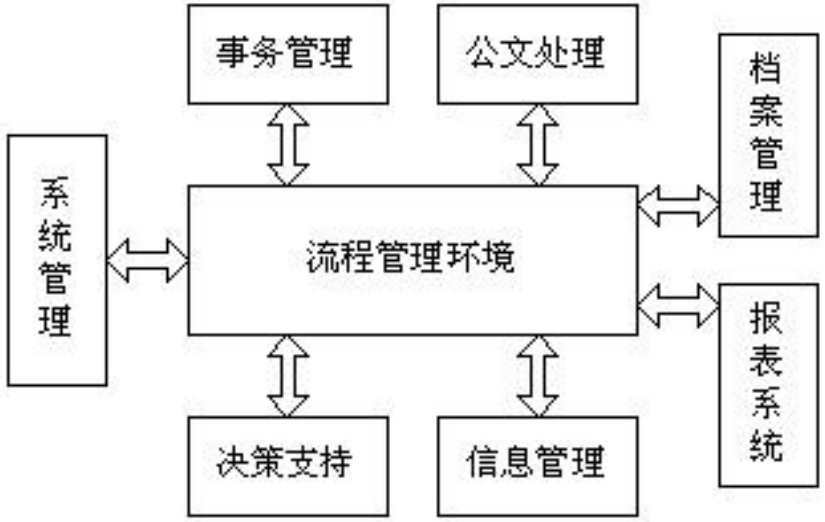


图1-5 领域软件产品设计框架

- 该工作流平台向产品最终用户，提供流程自定义工具，使用户可以自定义出所需要的工作业务流程，并可对流程流转过程进行实时监控。
- 向软件开发人员，提供了快速应用开发工具以及API，使开发人员只要调用该工作流平台API，就可以实现复杂流程业务程序。

1.2.5 构件重用实例

- 通过**系统构件的分层**，将频繁变动的业务逻辑层分离出来，实现通用类构件的完全重用。
- 为此，设计师采用了**层次式软件体系结构**，将产品的系统构件模型定义为4个层次，如图1-6所示。

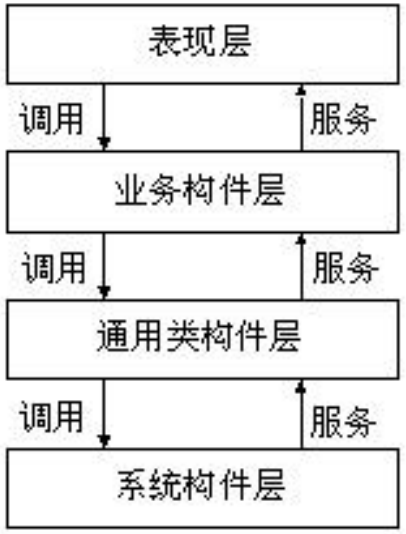


图1-6 层次式体系结构

1.2.5 构件重用实例

- （1）系统构件层。系统构件层指系统开发平台本身所提供的类库，包括Java JDK类库等。

- （2）通用类构件层。通用类构件层是产品重用的核心，它能实现产品领域横向的重用，和产品开发过程中的纵向层次结构的重用。
  - 这一层主要包含了工作流平台核心模块、组织管理模块、系统管理模块、数据库连接、通用打印和查询、权限验证和日期处理等与业务逻辑无关的类。

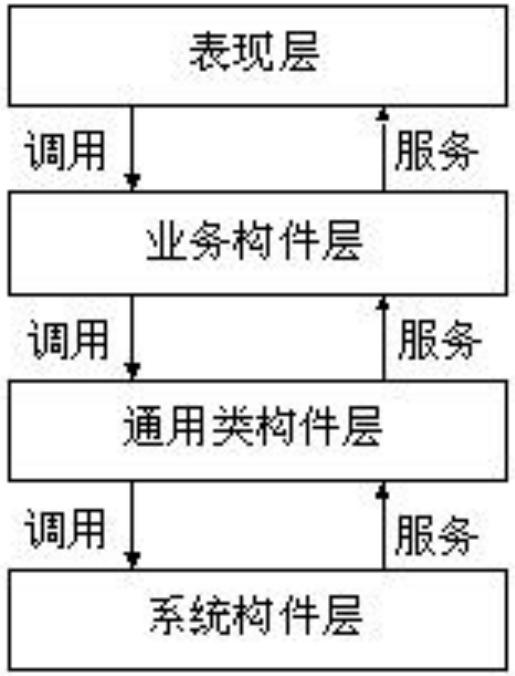


图1-6 层次式体系结构

1.2.5 构件重用实例

- （3）业务构件层。业务构件层指为了满足各个不同业务的需要，而设计的软件包，并在业务软件包中设置明确的接口，方便业务之间的交互，并可以实现系列产品之间的大粒度构件重用。

- （4）表现层。表现层主要采用JSP、Serverlet页面，来展现业务流程界面。
  - 在表现层中，JSP只调用JavaBean业务逻辑接口方法，作为用户与系统交互的接口，而不涉及任何业务逻辑。

图1-6 层次式体系结构

1.2.5 构件重用实例

- GOMS系统在层次式软件体系结构的基础上，利用面向对象的继承、封装和多态等特性，下层能够继承上层的所有功能，并可进行屏蔽、修改和扩充，从而实现功能的逐层扩展。
  - 通过抽象，能够将系统的大多数公共操作和通用的数据库表结构，提取出来，实现一个GOMS系统的基本操作库（基本类库）。
  - 通过封装，能够将完成某种功能的一系列操作和数据结构，封装在一个模块中，隐藏内部的具体实现过程。
  - 通过继承和重载，子类不但能够方便地获得、扩充或者修改父类的功能，而且还可以达到通过少量修改上层的方法，来实现软件的可扩展，从而解决管理流程不断变化、软件难以适应的问题。

1.2.5 构件重用实例

➢ 4、代码重用

- 在采用上述可重用的分析、设计方法后，系统的实现将变得相对容易。
- 在各代码段的实现时，只需要调用明确的接口，就可以实现处理功能，降低了算法的复杂度，大大提高了编码的效率及程序的可维护性。
  - ✓ 在编码时，主要采用了JavaBean和JSP技术，实现了业务实体逻辑和用户系统操作接口。
  - ✓ 在JavaBean中充分采用了Java的优秀的面向对象机制，实现了业务实体类的处理逻辑，并尽可能的达到了JavaBean方法的“一次编写，处处运行”的目标。
  - ✓ 在编写JSP时，完全引用了业务逻辑层的JavaBean，从而使JSP的页面编写，变得简单，并实现了业务逻辑的封装性，提高了JSP程序代码的安全性。

1.2.5 构件重用实例

- 另外，在编码过程中的一个重要的重用是操作（算法）的重用。
  - 由于在类设计时，基本上每一个类都提供了相似的功能，如新增、删除、修改、查询，而这些操作的算法基本上是一致的，差别只在于SQL语句的具体细节上。
  - 所以，在设计编码时，可以先设计一个基类提供这些操作，在其它类实现时，可以继承基类并重载或应用这些方法，实现操作算法的重用。

1.2.5 构件重用实例

➢ 5、组织结构的重用

- 在软件重用的过程中，仅仅有软件重用方法是不够的，还必须有重用的开发组织结构，予以支持。
- 建立了重用的组织框构，主要由3种成员组成：
  - ✓ 构件开发组
  - ✓ 构件应用组
  - ✓ 协调组。

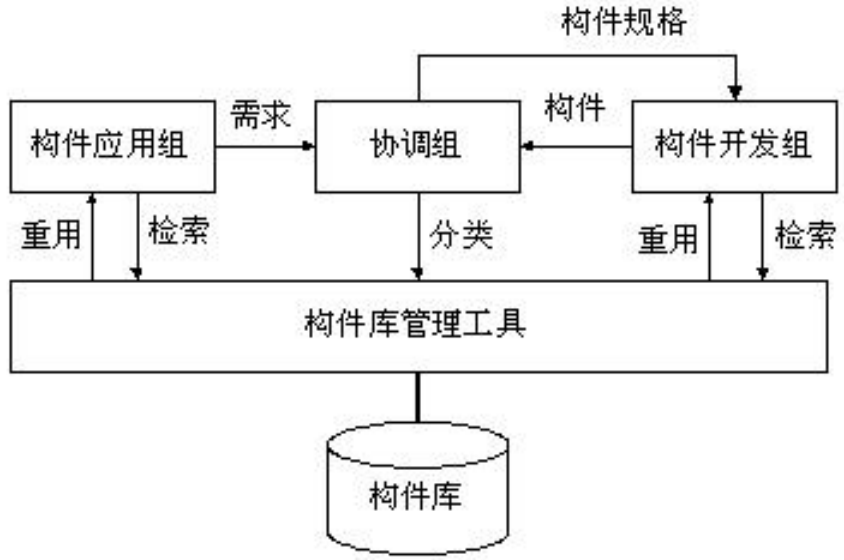


图1-7 组织结构关系

1.2.5 构件重用实例

- （1）构件开发组。
  - 构件开发组负责从协调组，接收构件设计规格说明，进行构件的设计和实现，把实现了的构件，交给协调组。
  - 构件开发组在接收到构件设计规格说明时，先在构件库中，检索是否已经存在满足规格说明的构件，或者相接近/类似的构件。
    - ✓ 如果存在这样的构件，则也可以直接进行重用，或者扩展已有功能以满足规格说明的需要。

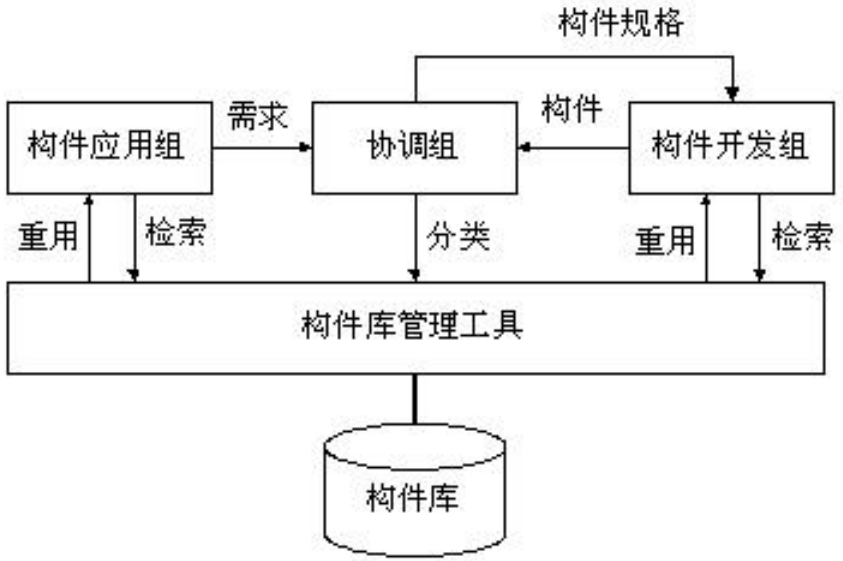


图1-7 组织结构关系

1.2.5 构件重用实例

（2）构件应用组

- 构件应用组负责业务逻辑的设计与实现。
- 在开发过程中，查询构件库，从中检索可重用构件，使用可重用构件，进行业务逻辑的快速实现。
- 如果构件库中没有需要的构件，则把构件的需求，交给协调组。

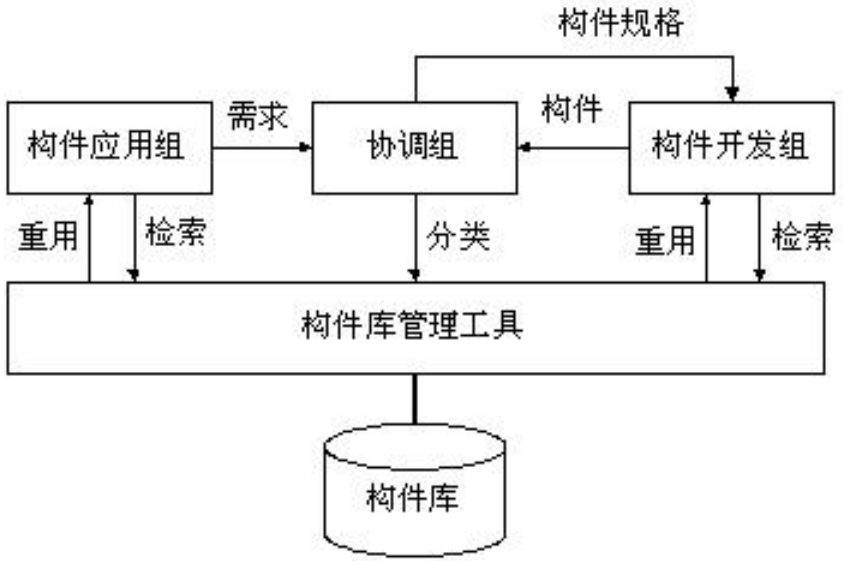


图1-7 组织结构关系

1.2.5 构件重用实例

➢ （3）协调组。

- 协调组负责在构件开发组和构件应用组之间起协调作用，接收并分析来自构件应用组的构件需求，然后就是自行进行构件开发，还是从第3方途径获取构件的问题，做出决策。
- 如果决定自行开发，就形成构件设计规格说明，交给构件开发组。
- 协调组同时负责构件的资格确认，质量控制，分类和存储工作。

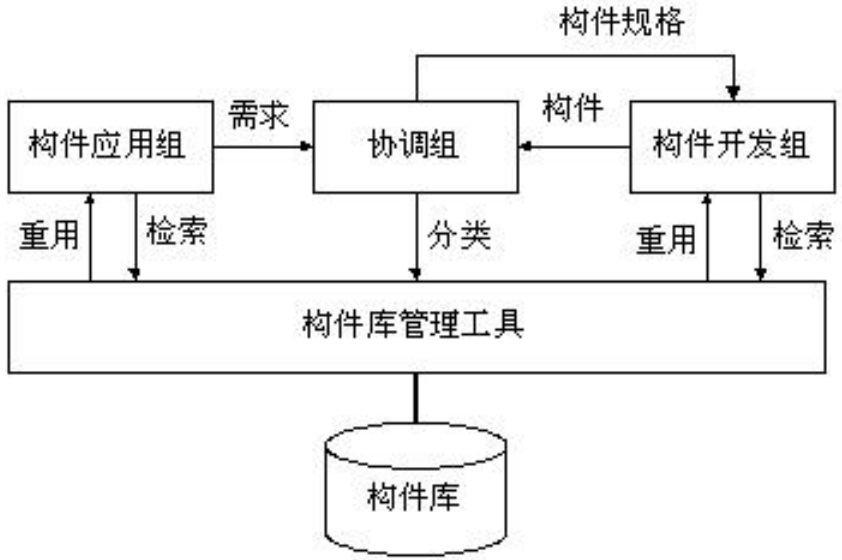


图1-7 组织结构关系

1.2.5 构件重用实例

➢ 7、小结

- 在通用办公管理系统及电子政务产品开发过程中，实现了产品领域横向的重用和产品开发过程中的纵向层次结构的重用。
- 在项目规划和整个软件开发过程中，系统地采用了重用的策略，并建立了一整套重用管理机制，从而大大提高了软件产品的可重用性和软件生产率，并为后继产品的开发提供了良好的可重用基础。

1.3 软件体系结构的兴起和发展

- 20世纪60年代的软件危机使得人们开始重视软件工程的研究。
- 起初，人们把软件设计的重点放在数据结构和算法的选择上，随着软件系统规模越来越大、越来越复杂，整个系统的结构和规格说明，显得越来越重要。
- 对于大规模的复杂软件系统来说，对系统的总体结构进行设计和规格说明，比起对计算的算法和数据结构的选择，已经变得明显重要得多。

- 在此种背景下，人们认识到软件体系结构（software architecture）的重要性，并认为对软件体系结构的系统、深入的研究，将会成为提高软件生产率和解决软件维护问题的有效途径。

1.3 软件体系结构的兴起和发展

- 软件体系结构借鉴了计算机体系结构和网络体系结构中很多宝贵的思想和方法。
- 软件体系结构研究的主要内容涉及：
  - 软件体系结构描述
  - 软件体系结构风格
  - 软件体系结构评价
  - 软件体系结构的形式化方法等

1.3.1 软件体系结构的定义

- （1）Mary Shaw和David Garlan认为软件体系结构是软件设计过程中的一个层次，这一层次超越计算过程中的算法设计和数据结构设计。
- 软件体系结构所关注的是：算法与数据结构之上，关于整体系统结构设计和描述方面的一些问题，如：
  - 全局组织和全局控制结构
  - 关于通讯、同步与数据存取的协议
  - 给设计元素分配特定功能，设计元素的组织，规模和性能
  - 设计元素的物理分布与合成
  - 设计方案的选择、评估与实现等。

1.3.1 软件体系结构的定义

- （2）David Garlan和Dewne Perry于1995年在IEEE软件工程学报上采用了如下的定义：
  - 软件体系结构是一个程序/系统各构件的结构、它们之间的相互关系，以及进行设计的原则和随时间演化的指导方针。

1.3.1 软件体系结构的定义

- （3）1997年，Bass，Ctements和Kazman在《Software Architecture in Practice》一书中给出如下的定义：
  - 一个程序或计算机系统的软件体系结构包括：
    - ✓ 一个或一组软件构件、软件构件的外部的可见特性及其相互关系。
    - ✓ 其中，“软件外部的可见特性”是指软件构件提供的服务、性能、特性、错误处理、共享资源使用等。

1.3.1 软件体系结构的定义

- 总之，软件体系结构为软件系统，提供了一个结构、行为和属性的*高级抽象*，由构成系统的元素的描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束组成。

1.3.2 软件体系结构的意义

- 软件体系结构是整个软件系统的骨架，在软件开发中起着非常重要的作用。

- 1、体系结构是风险承担者（stakeholder）进行交流的手段
  - 软件体系结构代表了系统的公共的高层次的抽象。
  - 这样，系统的大部分有关人员（即使不是全部）能把它，作为建立一个互相理解的基础，形成统一认识，互相交流。

1.3.2 软件体系结构的意义

- 因为软件系统的各个风险承担者（客户、用户、项目管理人员、设计开发人员以及测试人员等）关心着系统的各个不同方面，而这些方面都受体系结构的影响，所以体系结构可能是大家都关心的一个重要制品（即使是从不同愿望出发）。例如，
  - 用户，关心系统是否满足可用性和可靠性需求。
  - 客户，关心的是系统能否在规定时间内完成，并且开支是否在预算范围内。
  - 管理人员，担心在经费支出和进度条件下，按此体系结构，能否使开发组成员在一定程度上独立开发，各部分的交互是否遵循统一的规范，开发进度是否可控。
  - 开发人员，关心的是如何才能实现体系结构的各项目标
- 总之，体系结构提供了一种共同语言，来表达各种关注和协商，进而对大型复杂系统能进行理智的管理。这对项目最终的质量和使用有极大的影响。

1.3.2 软件体系结构的意义

➢ 2、体系结构是早期设计决策的体现

- 软件体系结构体现了系统的最早的一组设计决策（decision），这些早期的决策，比起以后的开发、设计、编码或运行服务及维护阶段的工作重要得多，对系统生命周期的影响也大得多。
- 早期决策的正确性更加难以保证，而且一旦这些决策发生改变，将对系统整体和全局的影响很大。

1.3.2 软件体系结构的意义

➢ （1）软件体系结构明确了对系统实现方案所施加的约束条件

- **系统实现，应严格遵循在体系结构设计中所做出的关于系统结构的决策。**（注：也就是说，系统的实现方案要遵循由体系结构设计决策所带来的各种要求，即约束条件）
- **在具体实现时，必须按照体系结构的设计方案，将系统分成若干个组成部分，各部分必须按照预定的方式进行交互，而且每个部分也必须具有体系结构中所规定的外部特征。**
- **这些约束是在系统级或项目范围内作出的，每个构件上工作的实现者是看不见的。**
  - ✓ **构件的开发人员在体系结构给定的约束下，进行开发。**
  - ✓ **软件体系结构的设计者不必是算法的设计者，只需重点考虑系统的总体权衡（tradeoff）问题**

1.3.2 软件体系结构的意义

- （2）软件体系结构决定了开发和维护团队的组织结构
  - 体系结构不仅规定了所开发软件的系统结构，还影响着项目开发组织的结构。
  - 开发一个大型系统时，通常采用的是基于系统的任务划分结构，即把系统的不同部分，交由不同的小组，去开发。
  - 体系结构中包含了对系统的最高层次的分解，因而一般被作为任务划分结构的基础。

1.3.2 软件体系结构的意义

- （2）软件体系结构决定了开发和维护组织的组织结构
  - **任务划分结构**还规定了履行任务的开发小组。
  - 各开发小组按照体系结构中对各主要构件接口的规定，进行交流。
  - 一旦进入维护阶段，维护活动也会反映出软件的结构，常由不同的小组分别负责对各具体部分的维护。

1.3.2 软件体系结构的意义

- （3）软件体系结构制约着系统的质量属性
  - 小的软件系统可以通过编程或调试措施，来达到质量属性的要求，而随着软件系统规模的扩大，这种技巧也将越来越无法满足要求。
  - 这是因为，在大型软件系统中，质量属性更多地是由系统结构和功能划分，来实现的，而不再主要依靠所选用的算法或数据结构。

1.3.2 软件体系结构的意义

（3）软件体系结构制约着系统的质量属性

- 系统的质量属性可以分为两类：
  - ✓ 第一类是可以通过运行软件，并观察其效果来度量的，如功能、性能、安全性及可靠性等；
  - ✓ 第二类是指那些无法通过观察系统来度量，只能通过观察开发活动或维护活动，来考察的特性，包括各种可维护性问题，如可适应性、可移植性、可重用性等（例如，可重用性依赖于系统中的构件与其他构件的联系情况）。
- 但是，软件体系结构并不能单独保证系统所要求的功能与质量。
  - ✓ 低劣的下游设计及实现，都会破坏一个体系结构的构架。
- 好的软件体系结构是高质量的必要条件，但不是充分条件。

1.3.2 软件体系结构的意义

- （4）通过研究软件体系结构，可能预测软件的质量
  - 可以使用对体系结构的评价，来预测系统未来的质量属性。
  - 一些体系结构评估技术，可使设计师全面地对按某软件体系结构开发出来的软件产品的质量及缺陷，做出比较准确的预测。

1.3.2 软件体系结构的意义

（5）软件体系结构使推理和控制变更更加地简单

- 在整个软件生命周期内，变更有3类：局部的、非局部的和体系结构级的变更。
  - ✓ 局部变更是最经常发生的，也是最容易进行的，只需修改某一个构件就可以实现。
  - ✓ 非局部变更的实现则需对多个构件进行修改，但并不改动体系结构。
  - ✓ 体系结构级的变更是指：会影响各部分的相互关系，甚至要改动整个系统。
- **一个优秀的体系结构应该能使变更简单易行。**
- 重要的是要确定何时必须变更，确定哪种变更方式的风险最小，评估变更的效果，以及合理安排所提出的变更的实施顺序和优先级等。**所有这些都需要深入地洞察系统的各部分之间的关系。**

1.3.2 软件体系结构的意义

- （6）软件体系结构有助于实施原型化方法
  - 一旦确定了体系结构，就可以对其进行分析，并构造可执行的原型。
  - 从两个方面有助于开发活动的顺利进行，
    - ✓ 一是可以在软件生命周期的早期阶段，明确潜在的系统性能问题
    - ✓ 二是在软件生命周期的早期阶段，就能得到一个可执行的实验系统。
  - 这两个方面都有助于减少项目开发的潜在风险。

1.3.2 软件体系结构的意义

- （7）软件体系结构可以作为培训的基础
  - 在对项目组新成员介绍所开发的系统时，可以首先介绍系统的体系结构，以及对构件之间如何交互，从而实现系统需求的高层次的描述，让项目新成员很快进入角色。

1.3.2 软件体系结构的意义

➢ 3、软件体系结构是可传递和可重用的模型

- 软件体系结构体现了一个相对来说比较小，又可理解的模型。
- 软件体系结构级的重用意味着：体系结构的设计决策，能在具有相似需求的多个系统的设计中重复使用。这比代码级的重用，有更大的好处。
- 通过对体系结构的抽象，可以使设计师能够对一些经过实践证明是非常有效的体系结构层的构件，进行重用，从而提高设计的效率和可靠性。
- 在设计过程中，设计师常常会发现，对一个体系结构中的构件稍加抽象，就可以将它应用到其他设计中去，这样会大大降低设计工作的工作量。

1.3.3 软件体系结构的发展史

- 20世纪70年代以前，尤其是在以ALGOL 68为代表的高级语言出现以前，软件开发基本上都是汇编程序设计，此阶段系统规模较小，很少明确考虑系统结构，一般不存在系统建模工作。

- 70年代中后期，由于结构化开发方法的出现与广泛应用，软件开发中出现了概要设计与详细设计，而且主要任务是数据流设计与控制流设计，因此，此时软件结构已作为一个明确的概念，出现在系统的开发中。

1.3.3 软件体系结构的发展史

- 20世纪80年代初到90年代中期，是面向对象开发方法兴起与成熟阶段。由于对象是数据与基于数据之上操作的封装，因而在面向对象开发方法下，数据流设计与控制流设计，被统一为对象建模，同时，面向对象方法还提出了一些其他的结构视图。例如，
  - OMT（Object Modeling Technology，对象建模技术）方法提出了功能视图、对象视图与动态视图（包括状态图和事件追踪图）；
  - Booch方法提出了类视图、对象视图、状态迁移图、交互作用图、模块图、进程图；
  - 统一建模语言（Unified Modeling Language，UML）则从功能模型（用例视图）、静态模型（包括类图、对象图、构件图、包图、组合结构图）、动态模型（通信图、顺序图、定时图、状态图、活动图、交互概览图）、配置模型（制品图、配置图），描述应用系统的结构。

1.3.3 软件体系结构的发展史

- 20世纪90年代以后则是基于构件的软件开发时代。它强调软件开发应该采用构件化技术和体系结构技术，要求开发出的软件具备很强的自适应性、互操作性、可扩展性和可重用性。

- 此时，软件体系结构已经作为一个明确的文档和中间产品，存在于软件开发过程中，同时，软件体系结构作为一门学科分支，逐渐得到人们的重视，并成为软件工程领域的研究热点。

1.3.3 软件体系结构的发展史

- 纵观软件体系结构技术的发展过程，从最初的“无结构”设计，到现行的基于体系结构的软件开发，可以认为经历了4个阶段：

1. 无体系结构设计阶段。以汇编语言进行小规模应用程序开发为特征。
2. 萌芽阶段。出现了程序结构设计主题，以控制流图和数据流图构成软件结构为特征。
3. 初期阶段。出现了从不同侧面描述系统的结构模型，以UML为典型代表。
4. 高级阶段。以描述系统的高层抽象结构为中心，不关心具体的建模细节，该阶段以Kruchten提出的“4+1”模型为标志。

1.4 软件体系结构的应用现状

➢ 1、软件体系结构描述语言

- 在提高软件工程师对软件系统的描述和理解能力中，虽然软件体系结构描述起着重要作用，但这些抽象的描述通常是非形式化的和随意的。
- 体系结构设计经常难以理解，难以适于进行形式化分析和模拟，缺乏相应的支持工具，帮助设计师完成设计工作。
- 为了解决这个问题，用于描述和推理的形式化语言得以发展，这些语言就叫做**体系结构描述语言**（Architecture Description Language，ADL），ADL寻求增加软件体系结构设计的可理解性和重用性。

1.4 软件体系结构的应用现状

- ADL是这样一种语言，系统设计师可以利用它所提供的特性，进行软件系统概念体系结构的建模。
- ADL提供了具体的语法，与描述体系结构的概念框架。
- ADL使得系统开发者能够很好地描述他们设计的体系结构，以便与他人交流，能够用相应的工具，对体系结构进行分析。

1.4 软件体系结构的应用现状

- 这种描述语言的目的就是提供一种形式化和规范化的体系结构描述，从而使得体系结构的自动化分析变得可能。
- 研究人员已经提出了若干适用于特定领域的ADL，典型的有C2、Wright、Aesop、Unicon、Rapide、SADL、MetaH、Weaves等。

- Shaw和Garlan指出，一个好的ADL的框架应具备如下几个方面的特点，即组装性、抽象性、重用性、可配置性、可分析性等。
- 在此基础上，Medividovic提出了一种ADL的分类和比较框架，详细分析了多种典型的ADL的优点与不足，对当时ADL的研究做了比较全面的总结，并为将来的ADL的开发提供了很有价值的参考建议。

1.4 软件体系结构的应用现状

➢ 2、体系结构描述构造与表示

- 按照一定的描述方法，用某种体系结构描述语言，对体系结构进行描述的结果，被称为体系结构的表示；而将描述体系结构的过程，称为体系结构的构造。
- 在体系结构描述方面，Kruchten提出的“4+1”模型是当前软件体系结构描述的一个经典范例。
- “4+1”模型由逻辑视图、开发视图、过程视图和物理视图组成，并通过场景将这4个视图有机地结合起来，比较细致地描述了需求和体系结构之间的关系。

1.4 软件体系结构的应用现状

2、体系结构描述构造与表示

- 而Booch从UML的角度，给出了一种由设计视图、过程视图、实现视图和部署视图，再加上一个用例视图构成的体系结构描述模型。
- Medividovic则总结了用UML描述体系结构的三种途径：
  - ✓ 不改变UML用法，而直接对体系结构建模。
  - ✓ 利用UML支持的扩充机制，扩展UML的元模型，以实现对体系结构建模概念的支持。
  - ✓ 对UML进行扩充，增加体系结构建模元素。

1.4 软件体系结构的应用现状

2、体系结构描述构造与表示

- IEEE于1995年成立了体系结构工作组，综合了体系结构描述研究成果，并参考业界的体系结构描述的实践，起草了体系结构描述框架标准IEEE P1471。

1.4 软件体系结构的应用现状

➢ 2、体系结构描述构造与表示

- Rational从资产重用的角度，提出了体系结构描述的规格说明（框架），该建议草案已经提交给OMG，可望成为体系结构描述的规范。
- IEEE P1471和Rational 的ADS，都提出了体系结构视点（viewpoint）的概念，并从多个视点，描述体系结构的框架。

- 但问题在于：一个体系结构应该从哪几个视点进行考虑？每个视点由哪些视图构成？各种视点应当使用哪种体系结构描述语言，以及采用哪些体系结构建模和分析技术等。

1.4 软件体系结构的应用现状

2、体系结构描述构造与表示

- 综上所述，虽然UML作为一个工业化标准的可视化建模语言，支持多角度、多层次、多方面的建模需求，支持扩展，并有强大的工具支持，确实是一种可选的体系结构描述语言，但是根据Medividovic给出的体系结构语言的框架，UML不属于体系结构描述语言的范畴。

- 事实上，判断一种语言是否适合用作体系结构描述语言的关键在于，它能否表达体系结构描述语言应该表达的概念与抽象，如果需要转化，其复杂性如何?

1.4 软件体系结构的应用现状

➢ 3、体系结构设计、分析与验证

- 体系结构设计有两大类方法：过程驱动方法和问题列表驱动方法。
  - ✓ 基于过程驱动的体系结构设计方法适用范围广，易于裁减，具备动态特点，通用性与实践性强。
  - ✓ 而问题列表驱动法的基本思想是**枚举设计空间，并考虑设计维的相关性，以此来选择体系结构的风格。**显然，该方法适用于特定领域，是静态的，并可以实现量化体系结构设计空间。
    - 如Allen博士的论文专门研究了用户界面类的量化设计空间，提出了19个功能维，26个结构维，62条设计规则。

1.4 软件体系结构的应用现状

3、体系结构设计、分析与验证

- **体系结构设计研究的重点内容之一就是体系结构风格或模式，体系结构模式在本质上反映了一些特定的元素、按照特定的方式组成一个特定的结构，该结构应有利于上下文环境（context）下的特定问题的解决。**

1.4 软件体系结构的应用现状

➢ 3、体系结构设计、分析与验证

- 体系结构模式分为两个大类：固定术语和参考模型。
- 已知的固定术语类的体系结构模型包括：
  - ✓ 管道过滤器、客户/服务器、面向对象、黑板、分层、对等模式（基于事件调用方法，隐式调用，基于推理模式）、状态转换、一些派生的固定术语类的体系结构模式，包括Gen Voca，C2和REST等；
- 而参考模型则相对较多，常常与特定领域相关，如编译器的顺序参考模型和并行参考模型、信息系统的参考模型、航空模拟环境系统的参考模型等等。

1.4 软件体系结构的应用现状

3、体系结构设计、分析与验证

- **体系结构分析的内容可分为：结构分析、功能分析和非功能分析。**
- **而在进行非功能分析时，可以采用定量分析方法与推断的分析方法。**
- **在非功能分析的途径上，则可以采用单个体系结构分析与体系结构比较的分析方法。**
  - ✓ **Kazman等人提出了一种非功能分析的体系结构分析方法，并运用场景技术，提出了基于场景的体系结构分析方法。**
  - ✓ **Barbacci等人提出了多质量属性情况下的体系结构质量模型、分析与权衡方法。**

1.4 软件体系结构的应用现状

3、体系结构设计、分析与验证

- 体系结构测试，着重于仿真系统模型，解决体系结构层的主要问题。
  - ✓ 由于测试的抽象层次不同，体系结构测试策略可以分为单元/子系统/集成/验收测试等阶段的测试策略。
  - ✓ 在体系结构集成测试阶段，Debra等人提出了一组针对体系结构的测试覆盖标准，Paola Inveradi提出了一种基于CHAM的体系结构语义验证技术。
- 应该说，体系结构分析、设计和验证已经取得了很丰富的研究成果。但是这些方法存在着一个普遍缺点：可操作性差，难于实用化，因此并没有取得很好的实践效果。

1.4 软件体系结构的应用现状

➢ 4、体系结构发现、演化与重用

- 体系结构发现指的是：如何从已经存在的系统中，提取出软件的体系结构。这属于逆向工程范畴。

1.4 软件体系结构的应用现状

- 由于系统需求、技术、环境、分布等因素的变化，而最终导致软件体系结构的变动，称之为软件体系结构演化。
- 软件体系结构的演化分为静态演化和动态演化两种
  - 静态演化是指设计阶段的体系结构的变化。
  - 动态演化是指运行时体系结构变化。
- 可以用多值代数，或图重写理论，来解释软件体系结构的演化。
  - Esteban等人通过研究系统的动态可配置特性，提出了电信软件体系结构动态修改的方案。

1.4 软件体系结构的应用现状

- 体系结构演化的动态性分为有约束的和无约束的，以及结构动态性和语义动态性。
  - Darwin和C2都直接支持结构动态性
  - CHAM，Wright和Rapide支持语义动态性
  - 在C2中定义有专门支持体系结构修改的描述语言AML
  - Darwin对体系结构的修改，则采用相应的脚本语言
  - CHAM是通过多值演算，实现系统体系结构的变换
  - Wright通过顺序通信进程CSP，描述构件的交互语义

1.4 软件体系结构的应用现状

- **体系结构重用属于设计重用，比代码重用更抽象。**
  - 由于软件体系结构是系统的高层抽象，反映了系统的主要组成元素及其交互关系，因而较算法更稳定，更适合于重用。
  - 产品线中的体系结构重用将更有现实意义，并具有更大的相似性。
  - 体系结构模式就是体系结构重用研究的一个成果，而体系结构参考模型则是特定域软件体系结构的重用的方法。
  - 重用技术作为软件工程领域倡导的有效技术之一，在基于构件与体系结构的软件开发时代，软件体系结构重用将是一个重要的主题。

1.4 软件体系结构的应用现状

➢ 5、基于体系结构的软件开发方法

- 在基于构件和基于体系结构的软件开发，逐渐成为主流的开发方法的情况下，已经出现了基于构件的软件工程方法。
  - ✓ 但是，对体系结构的描述、表示、设计和分析以及验证等内容的研究，还相对不足。
- 以构件重用为核心技术，覆盖并贯穿整个软件开发全生命周期，并形成具有通用性和基础性的软件工程方法学，还有很大的距离。

1.4 软件体系结构的应用现状

➢ 6、特定领域的体系结构框架

- 鉴于特定领域的应用相互间具有很多相似的特征，领域中那些经过严格设计的、并且已经被工程实践证明是成熟的软件体系结构，具有很大的可重用潜力。
  - ✓ Rick Hayers-Roth和Will Tracz分别对特定领域的体系结构DSSA （Domain Specific Software Architecture）给出了不同的定义。
  - ✓ 前者侧重于DSSA的特征，强调系统由构件组成，适用于特定领域，有利于开发成功应用程序的标准结构。
  - ✓ 后者侧重于DSSA的组成要素，指出DSSA应该包括领域模型、参考需求、参考体系结构、相应的支持环境或设施、实例化、细化或评估的方法与过程。
- 两种DSSA定义都强调了参考体系结构的重要性。

1.4 软件体系结构的应用现状

6、特定领域的体系结构框架

- 特定领域的体系结构是将体系结构理论，应用到具体领域的过程。
- DSSA的实例有：
  - ✓ CASE体系结构
  - ✓ CAD软件的参考模型
  - ✓ 信息系统的参考体系结构
  - ✓ 网络体系结构DSSA
  - ✓ 机场信息系统的体系结构和信息处理DSSA等。

1.4 软件体系结构的应用现状

6、特定领域的体系结构框架

- 软件体系结构充当一个理解系统构件和它们之间关系的框架，特别是那些始终跨越时间和实现的属性。这个理解对于现在系统的分析和未来系统的综合很有必要。
- 在综合的支持下，DSSA提供了建立系列产品的基础，以可预测的方式，利用领域知识构造和维护模块、子系统和系统。

1.4 软件体系结构的应用现状

➢ 7、软件体系结构支持工具

- 几乎每种体系结构设计方法都有相应的支持工具，如
  - ✓ Unicon，Aesop等体系结构支持环境
  - ✓ C2的支持环境ArchStudio
  - ✓ 支持主动连接件的Tracer工具等。

1.4 软件体系结构的应用现状

7、软件体系结构支持工具

- 支持体系结构分析的工具有：
  - ✓ 支持静态分析的工具
  - ✓ 支持类型检查的工具
  - ✓ 支持体系结构层次依赖分析的工具
  - ✓ 支持体系结构动态特性仿真工具
  - ✓ 体系结构性能仿真工具等。
- 但与其他成熟的软件工程环境相比，体系结构设计的支持工具还很不成熟，难以实用化。

1.4 软件体系结构的应用现状

➢ 8、软件产品线体系结构

- **一个产品线代表着一组具有公共的系统需求集的软件系统，它们都是：**
  - ✓ **根据基本的用户需求，对标准的产品线构架进行定制。**
  - ✓ **将可重用构件与系统独有的部分，集成而得到的。**
- **软件产品线是一个十分适合专业的软件开发组织的软件开发方法，能有效地提高软件生产率和质量，缩短开发时间，降低总开发成本。**
- **软件体系结构是形成完整软件产品线中的重要制品。**

1.4 软件体系结构的应用现状

➢ 9、建立评价软件体系结构的方法

- 不同类型的系统需要不同的体系结构，甚至一个系统的不同子系统，也需要不同的体系结构。体系结构的选择，往往会成为一个系统设计成败的关键。
- 但是
  - ✓ 怎样才能知道为软件系统所选用的体系结构是恰当的呢?
  - ✓ 如何确保按照所选用的体系结构能顺利地开发出成功的软件产品呢?
- 要回答这些问题并不容易，因为它受到很多因素的影响，需要专门的方法来对其进行评估。

1.4 软件体系结构的应用现状

- 9、建立评价软件体系结构的方法

- **目前，常用的软件体系结构评估方法主要有以下3个。**

1. **体系结构权衡分析方法（Architecture Tradeoff Analysis Method，ATAM）**
2. **软件体系结构分析方法（Software Architecture Analysis Method，SAAM）**
3. **中间设计的积极评审（Active Reviews for Intermediate Design，ARID）**

Approaches to Architecture

Academic Approach

- focus on analytic evaluation of architectural models
- individual models
- rigorous modeling notations
- powerful analysis techniques
- depth over breadth
- special-purpose solutions

Industrial Approach

- focus on wide range of development issues
- families of models
- practicality over rigor
- architecture as the big picture in development
- breadth over depth
- general-purpose solutions

1.4 软件体系结构的应用现状

- Carnegie Mellon leadership in software architecture includes:
  - Establishing the basic framework of architectural styles, similar to patterns.
  - Leading the development of formal techniques for architectural analysis.
  - Designing, teaching, and exporting the first course in software architecture.
  - Translating research results to a form that reaches practitioners effectively.
  - Developing and disseminating a set of architecture-centric methods and standards from the SEI.

1.4 软件体系结构的应用现状

- David Garlan is a Professor in the School of Computer Science at Carnegie Mellon University.

- His research interests include:
  - software architecture
  - self-adaptive systems
  - formal methods
  - cyber-physical systems
- In 2012 and 2013 Garlan was elected Fellow of the IEEE ("for contributions to software architecture") and of the ACM ("for contributions to the development of software architecture as a discipline in software engineering.").

1.4 软件体系结构的应用现状

- In 2017 David Garlan, Bradley Schmerl, and Mary Shaw received the Carnegie Mellon School of Computer Science’s Allen Newell Award for Research Excellence "for the development of software architecture as an organizing principle for large-scale systems, its influence on software practice, and its recent applications to cyber-physical systems and run-time failure analysis."
- In 2016 Shaw received the George R. Stibitz Computer & Communications Pioneer Award "for seminal & pioneering contributions to software architecture & computer science curricula"
- In 2014 Shaw received the National Medal of Innovation and Technology.
- In 2012, Jonathan Aldrich received the an ICSE retrospective "most influential paper" award for his paper on his Arch Java research in the 10th previous ICSE.

The End