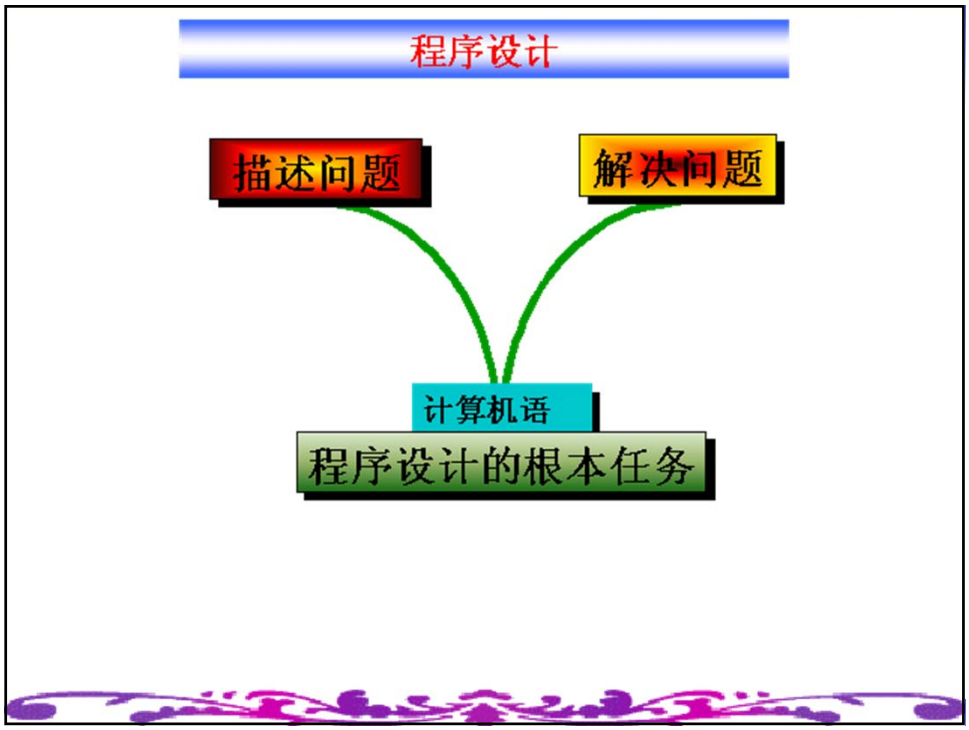
Programming Methodology
程序设计方法学

第6章　程序设计方法

主讲：刘悦
yliu@staff.shu.edu.cn



程序设计方法演进

- 手工艺式的程序设计方法
- 结构化程序设计
- 面向对象程序设计
- 面向Agent的程序设计
- 面向组件的程序设计
- 面向方面的程序设计

手工艺式的程序设计方法

程序设计就是对解决问题的算法以最终在计算机上能执行的程序代码的形式来描述的过程(也可先用程序设计语言来写,然后翻译到机器能执行的程序代码)。对第一代和第二代计算机来说,由于其速度较慢,存贮空间较小,加之大多数采用机器语言或汇编语言编写程序,所以要求程序的运行时间尽量短,占用的存贮单元尽量少。这样,程序设计是一项技巧性很强的工作,也可以说,采用的是一种手工艺式的设计方法,程序的可读性和可移植性较差,一度使用后即完成使命,寿命一般很短。

自60年代末到70年代初,随着计算机硬件技术的发展和计算机应用范围的不断扩大,需要研制一些大型的软件系统,如操作系统。而一个大的系统的编制周期较长,工作量很大,且由于传统的程序设计方法的局限性,设计出的软件系统往往隐藏着许多错误,这就给软件的使用和维护带来很大困难。一方面客观上需要研制大量的复杂的软件;另一方面,按照原有方法研制软件周期长,可靠性差,维护困难。这就产生了所谓的"软件危机"。

结构化程序设计…

延长程序的寿命,扩大其使用范围是克服软件危机的一种手段。要使许多人反复长期都能使用,就要考虑到程序必须是易读的,而且要适应环境的变化必须容易修正和维护。1968年,E.W.Dijkstra首先提出"GOTO语句是有害的",向传统的程序设计方法提出了挑战。他指出,GOTO语句使程序的静态结构与它的动态执行之间有很大的差别,造成程序难以阅读,难以查错。对一个程序来说,人们最关心它的正确与否,在去掉GOTO语句以后,可直接从程序结构上反映出程序执行的过程。

结构化程序设计方法是由E.W.Dijkstra在70年代首先提出的。它严格控制GOTO语句的使用,主张用顺序、选择和重复三种基本结构来嵌套连接成具有复杂层次的"结构化程序"。用这样的方法作出的程序在结构上有以下效果:

1)以控制结构为单位,只有一个入口,一个出口,所以能独立地理解这一部分;

2)能够以控制结构为单位,从上到下地顺序地阅读程序文本;

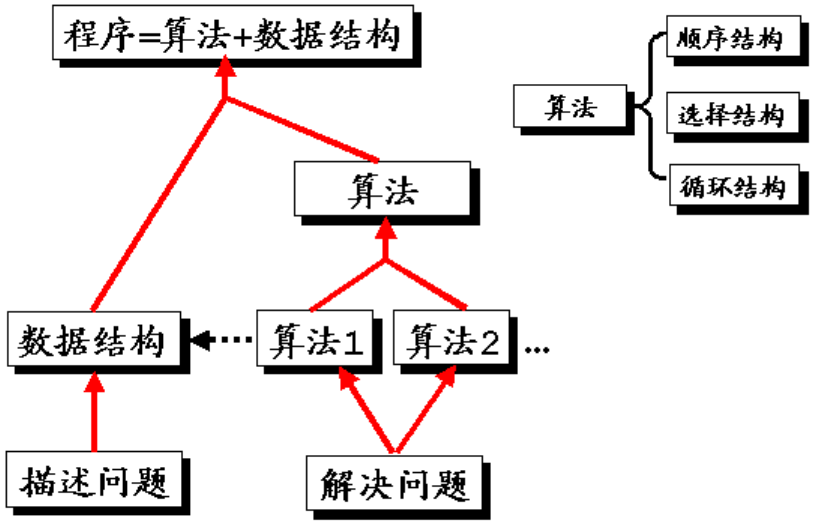
3)由于程序的静态描述与执行时的控制流程容易对应,所以能够方便地正确地理解程序的动作。

…结构化程序设计

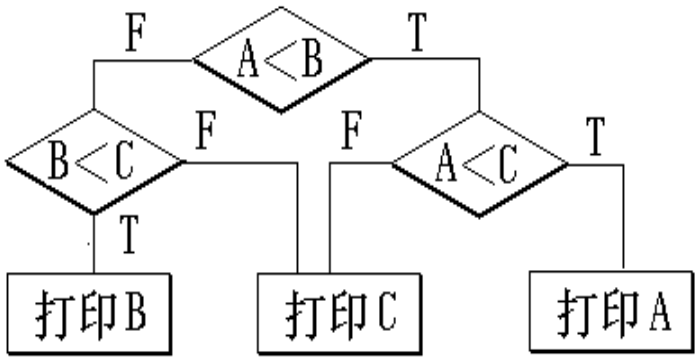
结构化程序设计方法支持自顶向下、逐步求精的设计思想。逐步求精的方法,就是在编制一个程序时,首先考虑程序的整体结构而忽视一些细节问题,然后逐步地、一层一层地细化程序直至选用的语言可以完全描述每一细节,即得到所期望的程序为止。结构化程序相比于非结构化程序有较好的可靠性、易验证性和可修改性。

结构化程序设计语言以Ada语言为代表,具有很强的表现能力,并能提高软件生产效率。在模块化的程序包、并行处理的任务、异常处理等方面都具有特色,它能使开发出的软件模块成为通用化的模块。

结构化程序设计方法



例1 打印A, B, C三数中最小者的程序



程序1

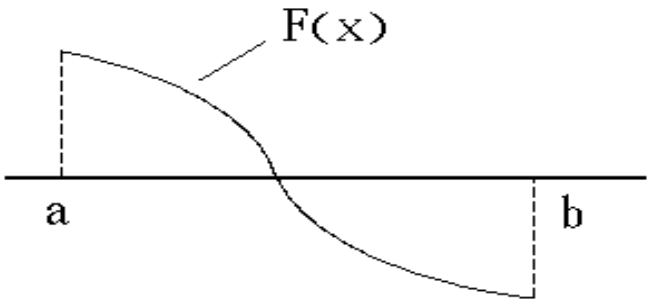
```
      if ( A < B ) goto 120;
      if ( B < C ) goto 110;
100   write ( C );
      goto 140;
110   write ( B );
      goto 140;
120   if ( A < C ) goto 130;
      goto 100;
130   write ( A );
140   end
```


程序2

```
if ( A < B ) and ( A < C ) then
   write ( A )
else
  if ( A ≥ B ) and ( B < C ) then
     write ( B )
  else
     write ( C )
   endif
endif
```

例2　用二分法求方程 f (x) = 0 在区间[a..b]中的根的程序

假设在闭区间[a..b]上函数 f (x) 有唯一的一个零点




```
f0 = f (a);   f1 = f (b);        //程序1
if  ( f0 * f1 <= 0 ) {
   x0 = a;   x1 = b;
  for  ( i = 1;  i <= n;  i++ ){
     xm = ( x0＋x1 ) / 2;   fm = f (xm);
    if ( abs (fm) < eps || abs (x1－x0) < eps)
       goto  finish;
    if  ( f0 * fm > 0 )
       { x0 = xm;   f0 = fm;  }
    else x1 = xm;
  }
```

```
finish：  printf ("\n The root of this
  equation is %d\n"，  xm );
}
```

- 单入口，两出口
- 正常出口是循环达到 *n* 次，非正常出口是循环中途控制转出到标号 *finish* 所在位置
- 可读性好


```
f0 = f (a);    f1 = f (b);    //程序2
if ( f0 * f1 <= 0 ) {
   x0 = a;  x1 = b;
   for ( i = 1;  i <= n;  i++ ) {   //正常出口
      xm = ( x0＋x1) / 2;  fm = f (xm);
      if (abs( fm )＜eps || abs( x1－x0 )＜eps)
          break;        //非正常出口
      if  ( f0 * fm＞0)
        { x0 = xm;  f0 = fm;  }
      else
         x1 = xm;
   }
}
```

```
f0 = f (a)；  f1 = f (b)；     //程序3
if ( f0 * f1 <= 0){
  x0 = a；  x1 = b；  i = 1；  finished = 0；
 while (i <= n && finished == 0) {
    xm = ( x0＋x1 ) / 2；  fm = f (xm)；
    if (abs( fm)＜eps || abs( x1－x0 )＜eps)
       finished = 1；
    if (finished == 0)
```



```
      if  ( f0 * fm＞0 )
          { x0 = xm；  f0 = fm；  }
        else
          x1 = xm ；
  }
}
```

- 引入布尔变量 ***finished***，改 for 型循环为 while 型，将单入口多出口结构改为单入口单出口结构。

面向对象程序设计…

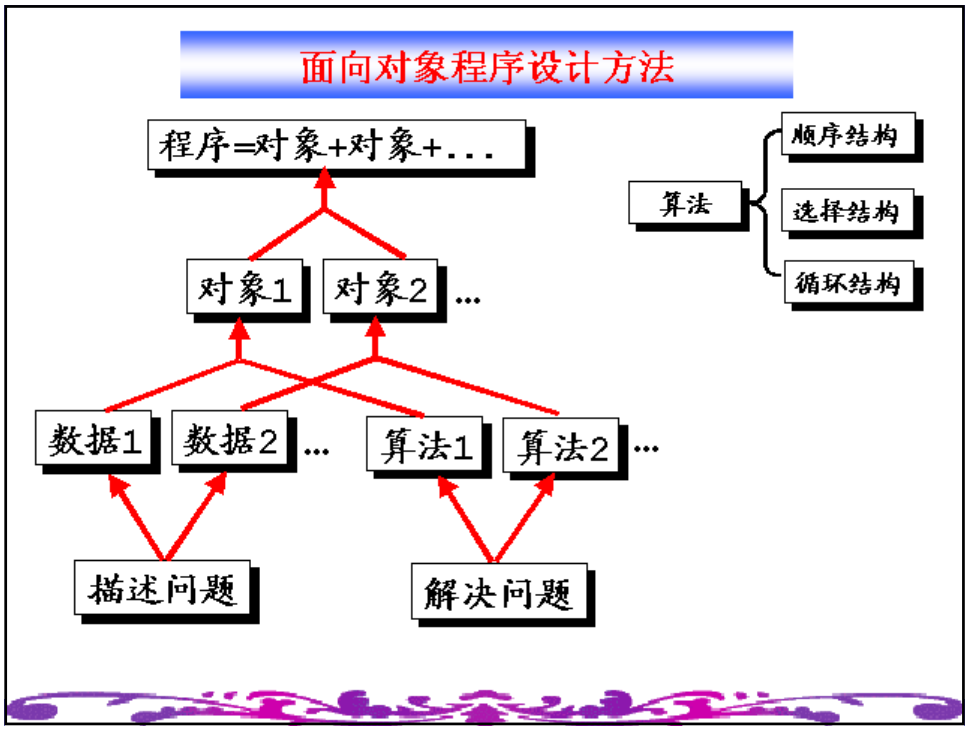
进入90年代，软件的研究和开发人员面临着不容乐观的形势。一方面，硬件的发展十分迅速，应用对软件系统在规模与复杂程序方面的要求在不断提高；另一方面，软件与硬件能力的差距还在继续拉大。人们普遍认为，传统的软件工具、软件技术和抽象层次越来越难以适应大规模复杂软件系统的开发特点。因此，软件能力问题已成为制约软件发展的主要因素。

面向对象程序设计方法是当今用于开发和扩充复杂软件系统的最有效的解决方法，它集数据抽象、抽象数据类型、类继承为一体，使软件工程公认的模块化、信息屏蔽、抽象、局部化、可重用性、可扩展性等原则在面向对象语言机制下得以充分实现。它将计算机环境视为一个对象集合体，而对象被定义为现实世界中实体的模型。它将数据以及对数据的操作组合在一起，这样，一个抽象数据类型、一个程序模块或硬件组成，甚至于计算机均可被视为一个对象。

…面向对象程序设计

面向对象程序设计的基础构件是对象和类，基本机制则是方法、消息和继承性。一个对象中的数据代表着它的状态，方法则代表了它的行为。外界要改变一个对象的状态，也就是对它所包含的数据进行操作，只能向这个对象发出请求（消息），然后由这个对象的对应方法来改变状态。这就是对象的封闭性。方法对外界只提供接口而不暴露细节，数据的表示（即数据结构）也对外屏蔽，这就是信息屏蔽。因此，对象呈现出抽象数据类型的特征。具有相同特性（指数据结构和方法）和共同用途的一组对象，在面向对象程序设计中被抽象成一个类。一个新引入的类可以从已有的类中继承特性，这就是继承性。这样程序员可以直接重用已有的类，所需的只是补充定义必要的特性。因此，面向对象程序设计把重用和抽象这两个概念结合在一起，成为增强软件能力的武器。

面向对象程序设计的特点很多，如建立在类及其继承性基础上的重用能力；可以应付复杂的大型的软件开发；便于扩展与维护，抽象程度高，因而具有较高的生产率。目前，面向对象程序设计语言如Java、C++等正在众多的程序设计语言中独领风骚。



面向Agent的程序设计...

在计算机科学主流中,Agent 的概念作为一个自包含、并行执行的软件过程,能够封装一些状态并通过传递消息与其它 Agent 通信,被看作是面向对象程序设计的一个自然发展。正如我们所知,Agent 意为系统中能够完成确立的一个或一系列任务的实体。根据 Agent 的用途,可将其划分为强 Agent 和弱 Agent。面向 Agent 的程序设计方法是一种"新的基于计算的社会观点的程序设计范式"。面向 Agent 的程序设计方法(AOP)与面向对象程序设计方法的最基本的区别在于 Agent 的社会性。这里的 Agent 可以包括对象实体,人的干预、外界环境等也同样可以视为 Agent。面向 Agent 程序设计的主要思想是:根据 Agent 理论所提出的代表 Agent 特性的精神的有意识的概念直接设计 Agent。这样一个目的后面的动机是,人们运用意愿姿态作为一个代表复杂系统的抽象机制。

Shoham 提出一个发展完全的 AOP 系统要有以下三个组成部分:

1)一个定义 Agent 的精神状态的逻辑系统;

2)一个设计 Agent 的解释的程序设计语言;

3)一个"Agent 化"过程,连接 Agent 程序成低级可执行系统。

…面向Agent的程序设计

如在 AGENT 0 语言中，系统中的逻辑成分是一个量化多模态逻辑，允许与时间直接关联。逻辑包含三个模态：信念、承诺和能力。与模态逻辑相应的是 AGENT 0 程序设计语言。在这种语言中，一个 Agent 被指定为一个能力集、一个初始信念和承诺集和一个承诺规则集。其中决定 Agent 如何行动的成分是承诺规则集，每个承诺规则集包含一个消息条件、一个精神条件和一个行为。为了决定这样一个规则是否适用，消息条件与 Agent 已收到的消息是相反匹配的；精神条件则针对 Agent 的信念而匹配。若规则适用，则 Agent 变得对行为承诺。行为可以是私有的，相当于一个内部执行的程序，或者是通信联络的，如发送消息。消息被限定为下述三种类型之一："要求"或"非要求"执行或制止行为和"通知"消息。要求和非要求消息典型地导致 Agent 的承诺更改；通知消息导致 Agent 的信念改变。

Agent 通常被赋予一些人类更经常使用的概念，如知识、信念、意愿和责任等，甚至于情绪。这样的 Agent 对于设计良好的人机界面以及一些智能化的人机交互系统是非常重要的。面向 Agent 的程序设计方法也可以说是智能化的程序设计方法，它使得程序设计在保留面向对象的优点的基础上，向着更为自然、更为智能的方向又跨近了一步。

面向组件的程序设计

- COM的起源
  - 源于OLE: Object Link and Embedding
    - OLE1 采用DDE(Dynamic Data Exchange) 在不同的程序之间进行通信
    - DDE缺点: 效率低,稳定性不好,使用不方便
    - COM是为克服上述不足而出现的
    - OLE2 以 COM 为基础，但OLE未能体现COM优点
- 什么是COM
  - 构件对象模型:Component Object Model
  - 客户与构件为了能够互操作而遵循的标准
  - COM标准包括规范与实现两部分
    - 规范部分：定义了构件之间的通信机制。这些规范不依赖于任何特定的语言和操作系统。
    - 实现部分：即COM库，为COM规范的具体实现提供一些核心服务。

COM构件

- COM构件：以 DLL或EXE形式发布的代码
  - 语言无关
  - 以二进制形式发布
  - 可以在不妨碍客户的形式下被升级
  - 可以透明地在网络上被重新分配
- 构件与类：一个构件可以由多个类实现
- 接口与类：一个类可以实现多个接口

COM库 (COM Library)

- 功能:
  - (1)实现客户方与服务器方COM应用的创建过程
  - (2)COM通过注册表查找本地服务器(即EXE程序) 以及程序名与CLSID的转换
  - (3)提供标准的内存控制方法
  - DCOM的实现提供了分布式环境下的通信机制
  - 在操作系统层次
  - 以DLL文件的形式存在

COM特性

- 语言无关性
  - 为跨语言合作开发提供了统一标准
  - 并得到不同集成开发环境的支持
- 进程透明性
  - 进程内服务程序: DLL
  - 本地服务程序: EXE
  - 远地服务程序: DLL或EXE
  - 实现进程透明性的关键是COM库
    - 它负责服务体的定位
    - 管理对象的创建及对象与客户之间的通信
- 复用性
  - 包含方式
    - 聚合方式

COM发展趋势

- 操作系统
  - 成为系统的基本软件模型
- 数据库
  - OLE DB/ADO 以 COM 的方式
  - 为数据访问提供一致的接口
- Internet
  - ActiveX包含了所有基于COM的Internet相关技术
- COM+
  - 增加MTS等服务

面向方面的程序设计…

- 传统的程序经常表现出一些不能自然地适合单个程序模块或者几个紧密相关的程序模块的行为。
  - 例如日志记录、对上下文敏感的错误处理、性能优化以及设计模式等等、我们将这种行为称为“横切关注点cross cutting concern）”，因为它跨越了给定编程模型中的典型职责界限。如果使用过用于横切关注点的代码，您就会知道缺乏模块性所带来的问日。因为横切行为的实现是分散的，开发人员发现这种行为难以作逻辑思维、实现和更改
  - 因此，面向方面的编程（Aspect－OrientedProgramming，AOP）应运而生
- AOP为开发者提供了一种描述横切关注点的机制，并能够自动将横切关注点织入到面向对象的软件系统中，从而实现了横切关注点的模块化．通过划分Aspect代码，横切关注点变得容易处理。开发者可以在编译时更改、插入或除去系统的Aspect，甚至重用系统的Aspect.

…面向方面的程序设计…

- 更重要的是，AOP可能对软件开发的过程造成根本性的影响。我们可以想象这样一种情况：OOP只用于表示对象之间的泛化一特化（generalization－specialization）关系（通过继承来表现），而对象之间的横向关联则完全用AOP来表现。这样，很多给对象之间横向关联增加灵活性的设计模式（例如Decorator、Role Object等）将不再必要
- 但是，现在的AOP还处于相当不完善的阶段：它只能应用于很少的几种语言环境下，并且必须掌握源代码才能进行织入．但以RUP之父Ivar Jacobson为代表的科学家们仍对AOP推崇备至：他认为AOP将最终改变整个软件开发的方式，并且更完美地实现“用例驱动”的开发思想.

…面向方面的程序设计…

- OOP不能很好地处理横越多个——经常是不相关的——模块的行为。相比之下，AOP填补了这个空白，它很可能会是编程方法学发展的一个里程碑。
- 把系统看作一批关注点，我们可以把一个复杂的系统看作是由多个关注点来组合实现的。
  - 一个典型的系统可能会包括几个方面的关注点，如业务逻辑、性能，数据存储、日志和调度信息、授权、安全、线程、错误检查等，还有开发过程中的关注点，如易懂、易维护、易追查、易扩展等，图1演示了由不同模块实现的一批关注点组成一个系统。

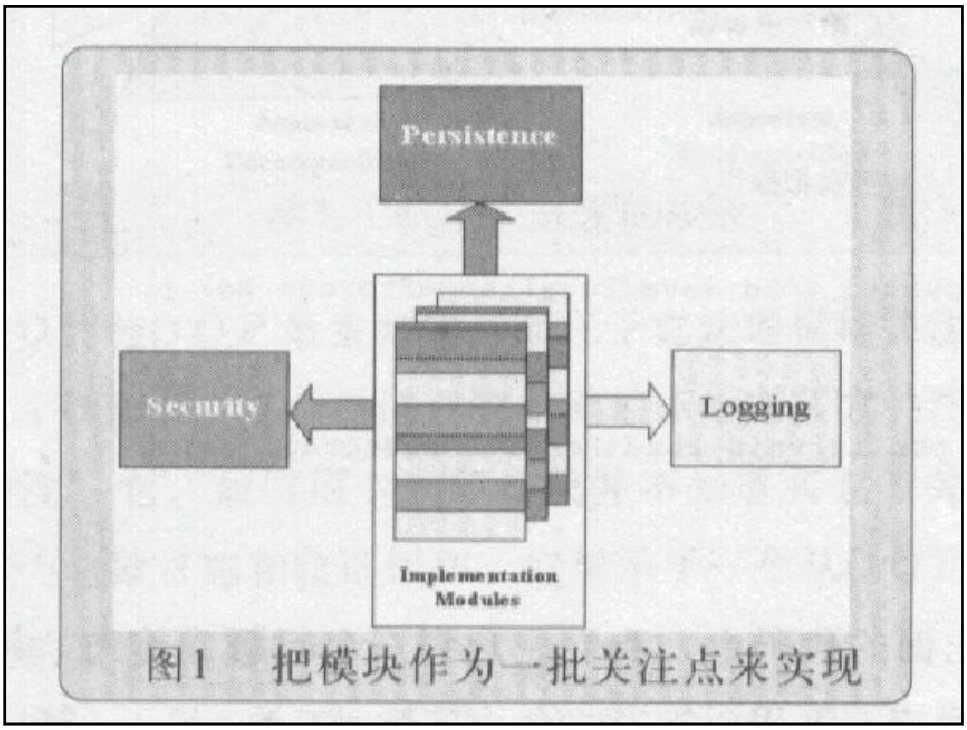


图1　把模块作为一批关注点来实现

…面向方面的程序设计…

- 开发人员建立一个系统以满足多个需求，我们可以大致地把这些需求分类为核心模块级需求和系统组需求。
  - 很多系统级需求一般来说是相互独立的，但它们一般都会横切许多核心模块。
  - 举个例子来说，一个典型的企业应用包含许多横切关注点，如验证、日志、资源地、系统管理、性能及存储管理等，每一个关注点都牵涉到几个子系统，如存储管理关注点会影响到所有的有状态业务对象。
  - 我们来看一个简单的例子，考虑一个封装了业务逻辑的类的实现框架：

…面向方面的程序设计

```
public class SomeBusinessClass extends otherBusinessClass
{
   //核心数据成员
   //其它数据成员：日志流，保证数据完整性的标志位等
   //重载基类的方法
   public void performSomeOperation (OperationInformation info){
   //安全性验证
   //检查传入数据是否满足协议
   //锁定对象以食品店当其他线程访问时的数据完整性
   //检查缓存中是否为最新信息
   //记录操作开始执行时间
   //执行核心操作
   //记录操作完成时间
   //给对象解锁
   }
   //一些类似操作
   public void save(PersitanceStorage ps){
   }
   public void save(PersitanceStorage ps){
   }
}
```

AOP开发步骤…

- AOP， 从其本质上讲，使你可以用一种松散耦合的方式来实现独立的关注点，然后 组合这些实现来建立最终系统、用它所建立的系统是使用松散偶合的，模块化实现的横切关注点来搭建的
- 与之对照 用OOP建立的系统则是用松散耦合的模块化实现的一般关注点来实现的
- 在AOP中，这些模块化单元叫“方面（aspect）”，而在OOP中，这些一般关注点的实现单元叫做类

… AOP开发步骤

- AOP 包括三个清晰的开发步骤：
  - 方面分解：分解需求找出横切关注点
    - 在这一步里，你把核心模块级关注点和系统级的横切关注点分离开来
    - 信用卡例子来说：可以分解出三个关注点：核心的信用卡处理、日志和验证
  - 关注点实现：各自独立的实现这些关注点
    - 还用上面信用卡的例子，你要实现信用卡处理单元、日志单元和验证单元
  - 方面的重新组合：在这一步里，方面集成器通过创建一个模块单元—一方面来指定重组的规则， 重组过程—一也叫织入（weaving）或结合（integrating）——则使用这些信息来构建最终系统
    - 还拿信用卡的那个例子来说，你可以指定（用某种AOP的实现所提供的语言）每个操作的开始和结束需要记录，并且每个操作在涉及到业务逻辑之前必须通过验证

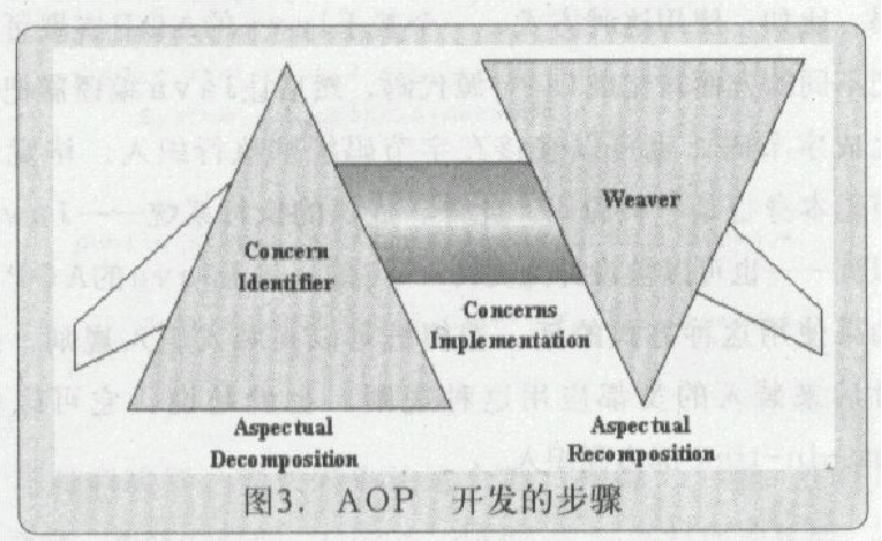


图3. AOP 开发的步骤

AOP与OOP

- AOP与OOP最重要的不同在于它处理横切关注点的方式
  - 在AOP中 每个关注点的实现都不知道其它关注点是否会“关注”它，如信用卡处理模块并不知道其它的关注点实现正在为它做日志和验证操作。它展示了一个从OOP转化来的强大的开发范型
  - 注意：一个AOP实现可以借助其它编程范型作为它的基础，从而原封不动的保留其基础范型的优点
    - 例如，AOP可以选择OOP作为它的基础范型，从而把OOP善于处理一股关注点的好处直接带过来。用这样一种实现，独立的一般关注点可以使用OOP技术、这就像过程型语言是许多OOP语言的基础一样

AOP语言

- 一种编程思想是否真正优秀，只有从实现语言上才能看到
  - 施乐公司帕洛阿尔托研究中心（Xerox PARC）开发了第一个AOP的开发环境——AsPectJ
  - 这个工具提供了一整套的语法，能够清楚地描述横切关注点，并将其织入到Java源代码中。织入后的代码仍是标准Java代码，因此AspectJ不会影响Java的移植能力
  - 此外，AspectJ提供了一个独立的IDE，并且能够嵌入到JBuilder等Java开发环境之中，无缝地提供AOP的能力
  - 关于AspectJ，可以在http://www.aspectj.org找到更多的信息

AOP的好处

- AOP可帮助我们解决上面提到的代码混乱和代码分散所带来的问题，它还有一些别的好处：
  - 模块化横切关注点：AOP用最小的耦合处理每个关注点，使得即使是横切关注点也是模块化的。这样的实现产生的系统，其代码的冗余小。模块化的实现还使得系统容易理解和维护
  - 系统容易扩展：由于方面模块根本不知道横切关注点，所以很容易通过建立新的方面加入新的功能。另外，当你往系统中加入新的模块时，已有的方面自动横切进来，使系统易于扩展
  - 设计决定的迟绑定：还记得设计师的两难局面吗？使用AOP 设计师可以推迟为将来的需求作决定，因为他可以把这种需求作为独立的方面很容易地实现
  - 更好的代码重用性：AOP把每个方面实现为独立的模块， 模块之间是松散耦合的
  - 举例来说，你可以用另外一个独立的日志写入器方面来替换当前的，用于把日志写入数据库，以满足不同的日志写入要求。松散藕合的实现通常意味着更好的代码重用性，AOP在使系统实现松散出合这一点上比OOP做得更好

程序的形式推导方法

- 基于最弱前置条件的概念，并用它定义我们的程序语言记号
- 原则：
  - 一个程序及其证明应同时开发，而且常由证明作引导
    - 因为要证明一个现有的程序是非常困难的，而较为有远见的做法是在程序编制的全过程中注入正确性证明的想法
  - 用理论提供启迪，在适当的地方使用常识及直觉，而碰到困难及复杂的事时由形式化的理论作支持
  - 绝不要由于某些基本的原则太明显而不予考虑，因为只有通过有意识地应用这些原则才能获得成功

例子...

- 写一个程序，对给定的整数x,y，要求将z置成x，y的最大者。
  - 这样我们就要有语句S，满足：{T} S {R:z=max(x,y)}
  - 在程序展开之前，R必须通过将max用其定义代替而精细化，得：R: $z \geq x \wedge z \geq y \wedge (z=x \vee z=y)$
  - 现在执行什么语句能够得到上式呢？因为上式中包含有z=x，故赋值语句z:=x是一种可能的语句。不妨先计算一下wp("z:=x",R),即
    - $wp("z:=x",R) = x \geq x \wedge x \geq y \wedge (x=x \vee x=y)$
      $= T \wedge x \geq y \wedge (T \vee x=y) = x \geq y$
    - 这表明在该条件下执行z:=x将建立R，故我们的程序的第一个版本可以是：if $x \geq y \rightarrow z:=x$ fi
    - 这个程序只要不夭折，就能做所希望的任务。由有关if语句的定理可知，要避免夭折，if语句的前置谓词Q必须蕴含条件的析取，即至少有一个条件在Q的初始状态中为真。但是，Q为T时，并不蕴含$x \geq y$ 。故至少还要一个条件子句。

…例子…

- 另一个可能建立R的方法是执行z:=y。从以上讨论显而易见y ≥x是所需的条件，即产生

  if x≥y → z:=x

  y ≥ x → z:=y

  fi

- 这样总有一个条件为真，因此这即为希望的程序。更形式化一些，我们可以用if定理来证明。

- 从这个例子可以看出，程序设计是一个面向目标的活动。这意味着，所要求的结果或目标R在程序的开发过程中起着比Q更为重要的作用。当然， Q也起作用，但一般很多启迪都来自R。程序设计的面向目标的特性也就是为什么用最弱前置谓词定义程序语义的理由之一。

…例子

- 为了说明这一点，我们暂时将Q搁置一边，并有一程序满足
  - {?} S {R:z=max(x,y)}
  - 只要S认为是完全的，即可检查Q⇒wp(S,R)。倘若相反，不考虑后置谓词R，而试图写一程序S，满足：{T} S { ？ }则每写一个S，检查是否有T⇒wp(S,z=max(x,y))或T⇒wp(S, R: z≥x∧z≥y∧(z=x∨z=y))。那么在找到一个正确的程序之前，你会写出多少个程序呢？

小结

- 试图解题前，要弄清楚问题到底是什么，即在编程序之前做到使前置和后置谓词准确而精细

模块化程序设计…

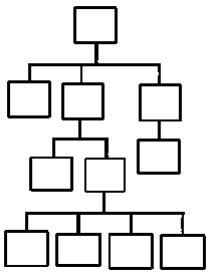
- “模块化程序设计” (modular programming), 是指“把系统或程序作为一组模块集合来开发的一种技术”
  - 目的：把一个复杂的任务断开成几个较小与较简单的子任务，它至少方便了正确的程序的编写
- 在大型软件设计中，系统设计的主要任务是确定系统由哪几部分组成，以及各部分之间的关系。软件系统结构设计由两个阶段组成：
  - 总体设计 (也称初步设计)：确定系统的模块化结构，即表现各模块之间的组成关系
    - 如何将一个系统划分为多个模块
    - 如何确定模块间的接口
    - 如何评价模块划分的质量
  - 详细设计
    - 对模块过程的说明，即软件模块内的过程设计

…模块化程序设计

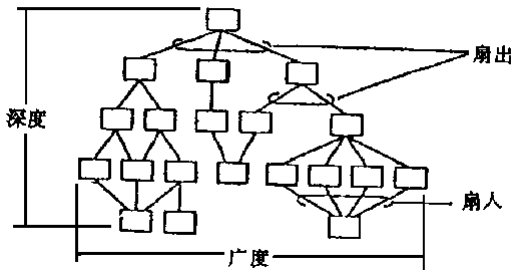
- 模块化设计的要领：能将一个系统分解成若干个容易管理的部分，使得每一部分可以被单独加工
- 模块化的一个基本问题是“应如何分解一个软件，以得到最佳的模块组合？”
- 模块化是所有程序的一种重要属性，软件模块化的核心问题是
  - 多级的模块程序结构
    - 自顶向下、逐步分解、求精和模块化的过程
  - 模块的独立性
    - 以内聚度和耦合度两个定性指标来衡量

软件的多级层次结构

- 软件结构是软件元素 (模块)间的关系表示
- 模块是软件结构的基础，软件结构的好坏完全由模块的属性体现出来



软件的结构描述



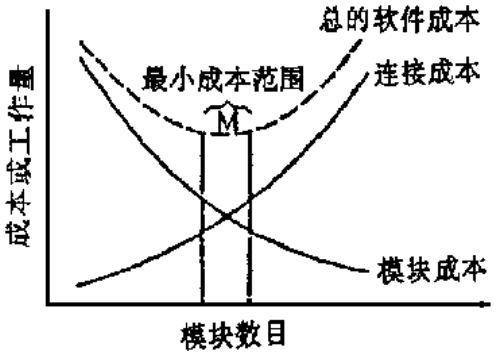
软件结构的度量

软件模块的独立性

- 希望每个模块只涉及该软件要求的一个具体的子功能，而且与软件结构其它部分的接口是简单的
- 提高模块的独立性，可使软件设计、调试、维护等过程变得容易和简单
- 模块独立性是用“内聚度”和“耦合度”这两个定性指标来度量的
  - 内聚度是度量一个模块功能的相对强度
  - 耦合度是度量模块之间的相互联系的程度
- 结构设计的目标是力求增加模块的内聚，尽量减少模块间的耦合，但内聚比耦合更为重要

软件模块化的价值

- 降低软件复杂性
- 便于软件设计
- 便于软件测试
- 提高软件的可维护性
- 建立公用 (标准)程序库实现程序共享



程序求逆

- 程序求逆从一个程序P推出另一个程序P^{-1}，它是计算P的逆，使反向执行程序
  - 即如果在执行P之后接着就执行P^{-1}，则就像什么也没有执行一样，即
  - $\{Q\}P; P^{-1}\{Q\}$

例子

- $P^{-1}(P(input))=input$
- x:=x+1的逆记为$(x:=x+1)^{-1}$，即为x:=x-1
- {x=3}x:=1的逆为{x=1}x:=3
- x:=x*x没有逆
- x,y:=y,x的逆即为其本身
- $(x:=x+y;y:=x-y;x:=x-y)^{-1}$ =x:=x+y;y:=x-y;x:=x-y

一般语句的求逆…

- 1、$(skip)^{-1}$ =piks
- 2、复合语句
  - $(S_1;S_2;..;S_n)^{-1} = S_n^{-1};S_{n-1}^{-1}; ..;S_2^{-1}; S_1^{-1}$
- 3、块结构程序
  - $(x:=C_1;S;\{x=C_2\})^{-1} = x:=C_2;S^{-1}\{x=C_1\}$

…一般语句的求逆

- 4、条件语句

$$\left(\begin{array}{l}\{B_1 \vee B_2\}\ \text{if } B_1 \rightarrow S_1 \{R_1\} \\ \qquad \square\ B_2 \rightarrow S_2 \{R_2\} \\ \qquad \text{fi } \{R_1 \vee R_2\}\end{array}\right)^{-1}$$

$$= \{R_2 \vee R_1\}\ \text{if } R_2 \rightarrow S_2^{-1}\{B_2\} \\ \qquad \square\ R_1 \rightarrow S_1^{-1}\{B_1\} \\ \qquad \text{fi } \{B_2 \vee B_1\}$$

- 5、循环语句
  - $\{\neg C\}\ do\ B \rightarrow S\ \{C\}\ od\ \{\neg B\}^{-1}$
  - $= \{\neg B\}\ do\ C \rightarrow S^{-1}\{B\}\ od\ \{\neg C\}$
  - 循环求逆时并不影响循环不变式

相关阅读材料…

6.1 程序设计方法演进

《浅谈程序设计方法的演进》

6.2逐步求精方法

清华大学本科软件工程讲义第六章：《程序编码》

6.3模块化程序设计方法

《模块化与计算机软件设计》

6.4结构化程序设计

清华大学本科软件工程讲义第六章：《程序编码》

…相关阅读材料

6.5面向对象的程序设计

清华大学本科软件工程讲义第八章：《面向对象技术》

《面向对象语言与过程语言比较之不足》

6.6面向Agent的程序设计

6.7面向COM的程序设计

北京大学高级软件工程讲义06：《构件对象模型:COM》

6.8程序的形式化推导方法

6.9程序求逆

作业5

- 以一个实例为例，简述前述程序设计方法或者自己熟悉的程序设计方法，突出其优、缺点
  - 上交形式：文档+程序
  - 检查形式：书面报告（电子文档）+ 口头报告