软件工程
第四部分 面向对象方法学
面向对象设计

11.1 面向对象设计的基本原则
11.2 系统设计(问题域/界面/任务/数据)
11.3 对象设计
11.4 设计模式 (专题报告,略)

11.1 面向对象设计的原则

- 面向对象设计分为系统设计和对象设计阶段。
  - ✓系统设计:
    - ①标识系统目标: 标识并区分各种质量属性的优先实现次序。
    - ②子系统分解: 根据用例模型和分析模型，将系统分解为一系列子系统。
    - ③子系统细化: 对各子系统不断分解求精，直到所有的设计目标都能满足为止。
  - ✓对象设计: 主要是追加一些与实现有关的对象，以及对已有对象进行细化。

11.1 面向对象设计的原则

设计原则1: 分治

- 软件系统分解为子系统
  - a) 分布式系统可以分解为客户机和服务器;
  - b) 系统可以分解为一系列子系统;
  - c) 子系统可以分解为一个或多个包;
  - d) 包可以分解为类;
  - e) 类可以分解为方法。

设计原则2: 尽可能增加内聚

- 不同内聚类型: 优先级从高到低排序
  1) 功能内聚: 构件只执行单一计算并返回结果，没有副作用。如函数过程。
  2) 层内聚: 相关服务放在一起，并有严格的层次结构，高层服务可访问低层服务，反之不可。如分层结构。
  3) 通信内聚: 访问或操作同一数据的过程放在一个类中，这些过程可以互相通信。如某个类设计。

4) 顺序内聚: 存在一系列过程，其中一个过程向另一个过程提供输入，这些过程放在一起，形成顺序内聚。如消息序列。
5) 过程内聚: 几个一次调用的过程放在一起，形成过程内聚。如调用结构。
6) 时间内聚: 程序执行过程中同一阶段内完成的操作放在一起，达到时间内聚。
7) 实用程序内聚: 逻辑上不能纳入其他内聚类型的相关实用程序放在一起，形成实用程序内聚。如可复用的过程或类。

设计原则3: 尽可能降低耦合

- 模块间存在相互依赖关系即为耦合。不同耦合类型从高向低排列有:
  1) 内容耦合: 一个构件修改另一个构件内部的数据，应始终避免。
  2) 公共耦合: 一组构件使用全局数据，就产生公共耦合。应通过封装降低公共耦合。
  3) 控制耦合: 一个过程通过标志、开关或命令显式地控制另一个过程的动作，就产生控制耦合。降低的方法是采用多态操作。

4) 标记耦合：在一个操作的参数表中将类作为参数，就产生标记耦合。可以改用传递简单变量或使用接口做参数降低耦合。
5) 数据耦合：在一个操作的参数表中用简单变量或简单的类（如string）作为参数，就产生数据耦合。应通过减少参数个数降低耦合。
6) 例程调用耦合：一个例程（或类操作）调用另一个例程，就产生例程调用耦合。如果出现例程调用序列，降低的方法是编写一个例程将这个调用序列封装起来。

7) 类型使用耦合：类将实例变量或本地变量声明为另一个类的实例，就产生类型（嵌套）使用耦合。可将变量的类型声明为包含所需操作的最通用的类或接口。
8) 包含/引入耦合：一个构件引入（import）一个包时就产生引入耦合，一个构件包含（include）另一个构件时，就产生包含耦合。
9) 外部耦合：模块对外部系统，如操作系统、共享库或硬件有依赖关系时就产生外部耦合。可通过信息隐蔽减少这种依赖关系。

设计原则4：尽可能提高抽象层次

- 设计应隐藏或推迟考虑细节以降低复杂性。
  - ✓类是包含过程抽象的数据抽象。
  - ✓使用泛化关系（父类）和接口，可进一步提高抽象层次。
  - ✓类中公有操作越少，抽象程度越高。
  - ✓类中所有变量都是私有的，抽象程度达到最高。
- 抽象可确保在设计时不必关心不必要的细节，能把握问题的本质并做出重要的决策。

设计原则5：尽可能提高可复用性

- 可以在算法、类、过程、框架和完整应用程序的级别上创建可复用性。
- 复用构件的机制包括过程调用和继承父类。

设计原则6：尽可能复用已有的设计和代码

- 复用已有的设计是对可复用性设计的补充。通过复用可从以往对可复用构件的投资中获益。

设计原则7：灵活性设计

- 积极预测将来可能在实现和功能上的变化，并采取相应措施。
- 在设计中引入灵活性的方法有：
  - ✓ 降低耦合并提高内聚（易于提高替换能力）
  - ✓ 建立抽象（创建有多态操作的接口和父类）
  - ✓ 不要将代码写死（消除代码中的常数）
  - ✓ 抛出异常（由操作的调用者处理异常）
  - ✓ 使用并创建可复用的代码

设计原则8：预计过期

- 积极预测将来可能在技术和运行环境上的变化，并为此采取相应措施。
- 在设计中应遵循的预计过期的规则有：
  - ✓ 避免使用早期发布的技术
  - ✓ 避免使用针对特定环境的软件库
  - ✓ 避免使用软件库中未编档的或很少使用的功能
  - ✓ 避免使用小公司或可能不提供长期支持的公司提供的可复用构件或特殊硬件
  - ✓ 使用众多厂商支持的标准语言和技术

设计原则9：可移植性设计

- 可移植性设计的主要目标是让软件在尽可能多的平台上运行。实现可移植性的规则有：
  - ✓避免使用特定环境的专有功能
  - ✓使用不依赖特定平台的程序设计语言
  - ✓小心使用可能依赖某一平台的类库
  - ✓了解其他语言可能依赖特殊硬件结构的功能和文本文件的差异

设计原则10：可测试性设计

- 设计时采取措施使得测试易于进行。
- 可测试性设计的最重要的方法是保证代码的所有功能都能脱离图形用户界面执行。

设计原则11：防御性设计

- 为提高可靠性，应确保不引入任何缺陷，能够处理其他代码不适当使用构件引起的问题。
- 按契约设计是防御性设计技术，其核心思想：
  - ✓被调用操作为正常执行必须满足的前置条件：调用操作在调用一个操作时有责任确保该操作的前置条件成立。
  - ✓被调用操作正常执行所得到的结果即为后置条件：要求被调用操作在返回前有责任保证这些后置条件成立。

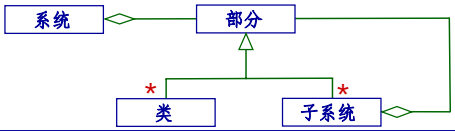
  - ✓ 被调用操作在执行时确保不会被修改的不变量(invariant)。
- 前置条件、后置条件和不变式都是布尔表达式，其计算结果为假，表示有错误发生。
- 可以使用断言机制。在重要构件的边界（如层）应始终保留严格的断言检测。

11.2 系统设计

- 系统设计的主要活动是进行子系统分解，并在此基础上定义子系统／构件之间的接口。
  - ✓首先，根据子系统可提供的服务来定义子系统（以后在对象设计中再根据子系统能提供的操作来定义子系统的接口），
  - ✓然后，对子系统细化，建立层次结构。要求对子系统的分解尽可能做到高内聚、低耦合。

11.2.1 系统设计的概念

1) 子系统和类
   - ✓ 在应用领域，为降低其复杂性，用类进行标识。在解域，为降低其复杂性，将系统分解为多个子系统，这些子系统又由若干表示解域的类构成。
   - ✓ 可以把子系统分解为更小的子系统。



- ✓ 在Java和Module 2中显式地提供了子系统的构造。在Java中采用了“包”，在Modula 2中采用了“模块”。C++没有显式地提供子系统的构造，可按问题需要对类分组。

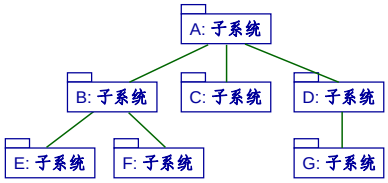
2) 服务和子系统接口

- ✓一个服务是一组有公共目的的相关操作。而子系统的特征是由该子系统提供给其他子系统的服务来刻画。
- ✓供其他子系统调用的某个子系统的操作集合就是子系统的接口。子系统的接口包括操作名、操作参数类型及返回值。
- ✓系统设计注重每个子系统提供服务的定义，即枚举所有的操作、操作参数和行为。

- ✓ 当编写子系统接口的文档时，不应涉及子系统实现的细节，其目的是减少子系统之间的依赖性，希望一旦需要修改子系统实现时，降低由于子系统变更而造成的影响。

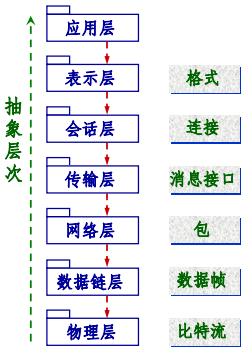
3) 子系统分层和划分

- ✓ 子系统分层的目的是建立系统的层次结构。每一层仅依赖于它下一层提供的服务，而对它的上一层可以一无所知。
- ✓ 下图是一个三层的系统结构。

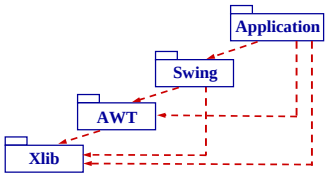


- ✓ 在层次结构中，子系统A、B、E构成了一个称之为垂直切片的系统分解子集。
- ✓ 如果每一层只能访问与其相邻的下一层，则称之为封闭体系结构；如果每一层还可访问比其相邻下一层更低的层次，则称之为开放体系结构。

- ✓封闭体系结构的例子就是开放系统互联参考模型，它由七层构成。每一层负责执行一个已预先定义好的协议功能。每一层都为其上一层提供服务，使用其低层的服务。
- ✓封闭体系结构的子系统之间满足低耦合，但产生速度和存储管理的问题，会导致某些非功能属性难以满足。



✓ 开放体系结构的一个例子是Java的Swing用户接口包。它允许人们绕过高层直接访问低层接口以克服性能瓶颈。



✓划分是将系统分解为独立的子系统，每个子系统负责某一类服务。

- 例如，一个车辆管理所的管理信息系统分为车管所组织机构管理、车辆管理、车主管理和法律事件管理等 4 个子系统。
- 每一个子系统对其他子系统的依赖度很低。

✓分解子系统时，首先进行划分，将一个系统分成几个高层的子系统，每个子系统负责一种功能，或运行在某特定的硬件节点上。

✓再将各子系统分层处理，分解成层次更低的小子系统。

✓过度分解会导致子系统之间接口的复杂化。

11.2.2 系统设计的目标

- 明确设计目标，是系统设计的第一步。有一些设计目标是从用户那里得到，更多的设计目标是从非功能需求或应用领域中抽取出来。
- 设计目标可以从以下 5 组设计准则中选择。
  - 性能、可靠性和最终用户准则往往从需求中明确地获取，也可以从应用领域中获取；
  - 成本和维护准则由客户和供应商提出。
- 将设计目标清楚地陈述，可以根据设计准则，做出重要的设计决策。

性能准则

设计准则	定　义
响应时间	用户提交请求后多长时间可得到系统的确认
吞吐量	在一段固定时间内系统能够处理多少任务
内存	系统运行需要占用多少内存空间

可靠性准则

设计准则	定　义
鲁棒性	面对非法用户进入后系统的存活能力
可靠性	明确期待的行为与观察到行为之间的差别
可用性	系统能够用来完成正常任务的时间百分比
容错性	在出错条件下系统的操作能力
保密性	系统忍受恶意攻击的能力
安全性	再出现错误时系统避免威胁到人类生命的能力

成本准则

设计准则	定　义
开发成本	开发系统初始版本的成本
部署成本	安装系统并培训用户的成本
升级成本	从原有系统中导出数据的成本，此准则导致了向后兼容性需求
维护成本	需要进行错误修复和增强系统的成本
管理成本	需要对系统进行管理的成本

最终用户准则

设计准则	定　义
效用	系统支持用户工作的困难程度
易用性	用户使用系统的困难程度

维护准则

设计准则	定　义
可扩展性	向系统中添加功能或新类的困难程度
可修改性	更改系统功能的困难程度
可适应性	将系统发送到不同应用领域的困难程度
可移植性	将系统移植到不同平台的困难程度
可读性	通过阅读源代码来理解系统的困难程度
需求的可追踪性	将代码映射到特定需求上的困难程度

- 在定义设计目标时，希望开发一个既安全可靠，又廉价的系统是不现实的，因此开发人员应当对所有可能的设计目标进行权衡，对必须的设计目标赋予优先级别。
- 一旦有了清晰的设计目标，就可以开始系统的初始分解。

权　衡	基 本 原 则
空间与速度	如果软件的响应时间或吞吐量不满足需求，则可以使用更多的存储空间来加快软件的执行速度。如果软件太大，则可以牺牲一定的速度对数据进行压缩处理。

权　衡	基 本 原 则
交付时间与功能	如果开发进度滞后于计划，则按时交付的功能可以少于预定交付的功能，或推迟交付所有功能。契约软件通常更强调功能，而商业外购软件则更强调交付日期。
交付时间与质量	如果测试滞后于计划，则可以按时交付带有错误的软件，或推迟交付带有少量错误的软件。
交付时间与人员配置	如果开发进度滞后于计划，可以在项目中增加资源以提高生产率。在多数情况下，这种选择只适用于早期项目。新的人员要经过培训方可使用。这样中途增加资源通常会降低生产率，还会增加软件开发的成本。

11.2.3 子系统分解

- 软件体系结构设计的第一步是做子系统分解，从而搭建软件体系结构的架构。
- 在系统设计中寻找子系统，与在分析过程中寻找对象的情况类似。在面向对象分析中介绍的Abbotts启发式准则，同样适合于子系统的识别。
- 初始子系统应该从需求的功能模型（用例模型）中导出。这方面有多种分解方案。
- 例如，Coad & Yourdon基于MVC模型，将系统大致分为问题领域、人机交互、任务管理、数据管理等 4 个子系统。

11.2.4 问题领域子系统

- 首先复制分析模型为设计模型中的问题领域部分，然后根据设计目标，对该部分进行修改。

❶可复用的设计／编程方面的类

- ✓ 根据需要，从类库选择可复用类、商业外购构件和遗留构件，把它们增加到问题解决方案中去。
- ✓ 标明复用类或构件中不需要的属性和操作。
- ✓ 增加从复用类（构件）到分析类之间的泛化关系，继承复用类（构件）的属性和方法。

- ✓ 把分析类中因继承复用类（构件）而成为多余的属性和操作标出。
- ✓ 修改分析类的结构和连接。

❷ 加入泛化类以建立类间协议

- ✓ 有时某些分析类要求一组类似的服务。此时，以这些分析类作为子类，定义一个父类。该父类定义为所有这些子类共用的一组服务名，作为公共的协议，用来与数据管理或其他外部系统部件通信。这些服务都是虚函数。在各个子类中定义其实现。

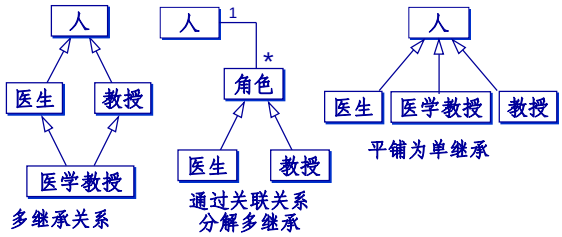
❸ 对泛化关系进行调整

✓ 在分析模型中可能包括有多继承关系，但实现时使用的程序设计语言可能只有单继承，甚至没有继承机制，这样就需变更问题领域部分中类的层次结构。

✓ 针对单继承语言的调整

➢ 把子类对象看做是父类对象所扮演的角色，通过关联关系把多继承的层次结构转换为单继承的层次结构。

➢ 把多继承的层次结构平铺，成为单继承的层次结构。在这种情况下，有些属性或操作在同层的子类中会重复出现。



✓ 针对无继承语言的调整

➢ 当使用无继承的编程语言时，必须把具有泛化关系的类层次平铺开，成为一组对象。

❹ 修改设计以提高性能

✓ 提高执行效率是系统设计的主要目标之一。有时必须改变问题领域的结构以提高效率。

✓ 如果类之间经常需要传送大量消息，可合并相关的类，使得通信成为对象内的通信，而不是对象之间的通信，或者使用全局数据作用域，打破封装的原则，以减少消息传递引起的速度损失。

✓ 增加某些属性到原来的类中，或增加低层的类，以保存暂时结果，避免每次都要重复计算造成速度损失。

✓为提高性能，在对分析模型进行大规模的改动之前，应考虑下面一些问题:

➢ 不要认为象C++之类的OOPL就一定效率高。事实表明，非OOPL的紧凑代码的效率比OOPL的效率高近10倍，但用非OOPL编程会令程序员非常疲劳，容易出错。

➢ 提高一个现存系统的工作效率比重新设计一个高效的系统要容易。一开始应当建立一个原始的简单的设计，实现和调试不会太困难。如果对设计有性能要求，只需加入少量的工作就可以了。

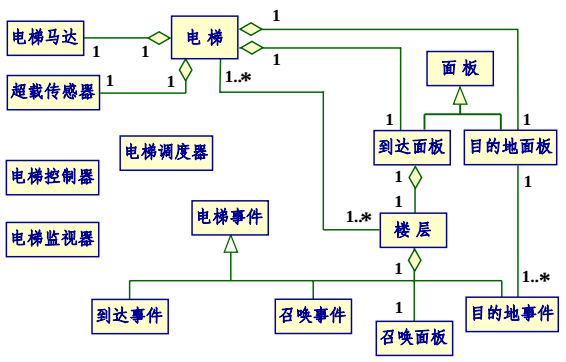
➢ 通常系统80%的开销都集中在20%的代码段上。与其为了尽量处处节省系统开销而破坏完善的系统结构，还不如找出系统开销最集中的地方，只对那部分做优化。

➢ 预测软件开销集中在什么地方是困难的。进行优化最有效的方法是在系统运行时使用性能监测工具对系统进行观测。一些像继承、动态绑定、消息传递等处理虽然看起来简单，但需要大量的系统开销。在代码复杂性与运行的低效之间没有相关性。

➢ 提高性能最好的方法是采用好的解决方案，而不是拼命地去节省几个微秒、几个字节。

电梯控制系统ECS的问题领域子系统

- 决定使用一个中央控制器（电梯控制器）控制和协调电梯的所有动作，包括解决电梯每到一个楼层减速问题。
- 增加“电梯监视器”，用一个独立的对象来执行监测功能（性能、安全性等）。在提交给实现者的规格说明中，将这个类放在一个 与ECS的主处理器分离的处理器上.
- 每个类增加服务Self Test来增强每个类的性能。在运行时执行有必要验证其功能的操作，并向电梯监视器报告。



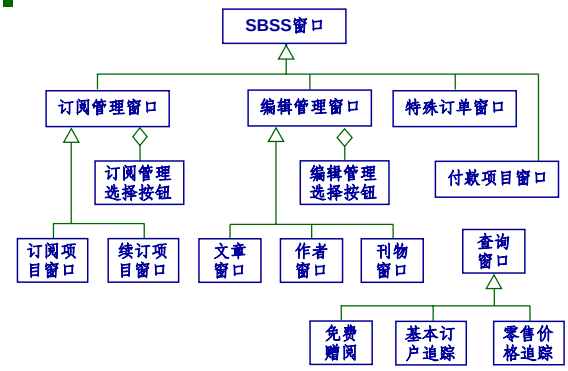
11.2.5 人机交互子系统

- 设计一个良好的用户界面是成功地实现一个软件系统的关键。在开发分析模型时，有意避开了如窗口、屏幕等依赖于实现的细节，目的是让系统规格说明独立于实现。
- 人机交互子系统在系统行为和用户交互的实现之间架起了一座桥梁。例如，可用 GUI 实现系统的用户界面。但当用户提出来改用语音对答式交互时，应当使这种改变尽可能容易。为此，只需将 GUI 式人机交互子系统替换成语音对答式人机交互子系统即可，系统其它部分都应保持不变。

例如，某杂志发行系统SBSS的人机接口

❶ 设计用户界面

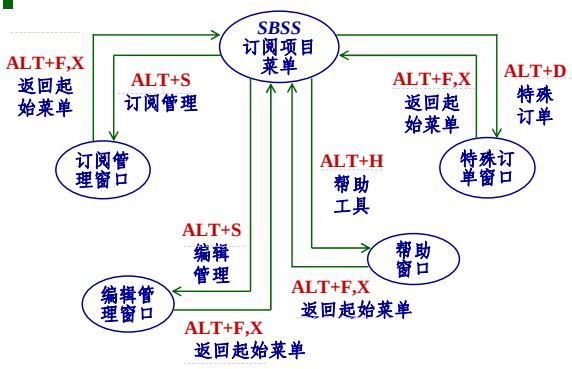
- ✓ SBSS的用户界面是一个传统的GUI。其中有许多窗口对象。但在用户与系统进行交互期间，只能有一个窗口呈现在用户面前。
- ✓ 通过使用聚合关系，窗口对象可以进一步分解为各种文本域、选择按钮、图符等。
- ✓ 如果使用某种 GUI 构筑软件包，只需对软件包输入合适的参数，该软件包就能提供所有的文本域、选择按钮、图符等，将这些细节作为窗口的属性就可以了。



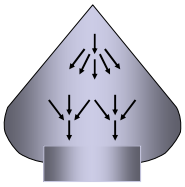
- ✓ 只要描述清楚所需要的窗口，这些窗口的导航细节，详细的窗口成分，我们就可以用现有的技术和工具来实现它们。

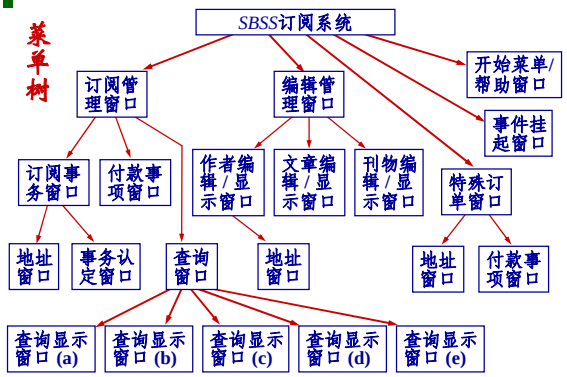
❷ 其他描述人机交互模型的工具

- ✓ 在描述复杂的菜单系统（GUI窗口）的拓扑结构时，常使用状态图或菜单树。
- ✓ 用于SBSS的单层状态图：
  - ➢ 每一个状态是GUI的一个窗口
  - ➢ 每一个迁移代表了窗口之间的切换
  - ➢ 与每个迁移相关联的是一个条件 / 动作对
  - ➢ 条件代表了用户选择，它导致迁移的发生
  - ➢ 动作代表了迁移发生时产生的请求



- ✓ 菜单树可以让用户直观地看到如何在界面上巡航。用菜单树表示人机交互的基本结构，有助于全局用户界面的可视化。例如，可以了解有多少种不同的方式能够访问“地址”的窗口。
- ✓ 用户界面可尽量采用清真寺式的结构：在界面的上层，希望扇出大于扇入，表明用户可以有多个可供选择的选项。在界面的低层，希望扇入大于扇出，表明一个单独的界面可以被多个双亲使用。





- 人机交互子系统的设计过程

1) 用户分类
   - ✓ 按技能层次分类：外行／初学者／熟练者／专家
   - ✓ 按组织层次分类：行政人员／管理人员／专业技术人员／其它办事员
   - ✓ 按职能分类：客户／职员
2) 描述人及其任务的脚本

对以上定义的每一类用户，考虑以下问题：谁、目的、特点、成功的关键因素、熟练程度以及任务脚本。

在OOATOOL™ 中有一个例子：

- ✓ 谁：分析员
- ✓ 目的：要求一个工具来辅助分析工作 (摆脱繁重的画图和检查图的工作)。
- ✓ 特点：年龄: 42岁；教育水平: 大学；限制: 不要微型打印，小于9个点的打印太小。
- ✓ 成功的关键因素：工具应能使分析工作顺利进行；工具不应与分析工作冲突；工具应能捕获假设和思想，能适时做出折衷；应能及时给出模型各个部分的文档。

- ✓ 熟练程度：专家。
- ✓ 任务脚本：
  - ☞ 主脚本：
    - 识别“核心的”类和对象；
    - 识别“核心”结构；
    - 在发现了新的属性或操作时随时都可以加进模型中去。
  - ☞ 检验模型：
    - 打印模型及其全部文档。

3) 设计命令层

- ✓ 研究现行的人机交互活动的内容和准则。这些准则可以是非形式的，如“输入时眼睛不易疲劳”，也可以是正式规定的；
- ✓ 建立一个初始的命令层。可以有多种形式，如一系列 Menu Screens、或一个 Menu Bar、或一系列Icons.
- ✓ 细化命令层。考虑以下几个问题。
  - ➢ 排列命令层次。把使用最频繁的操作放在前面；按照用户工作步骤排列。
  - ➢ 通过逐步分解，找到整体－局部模式，以帮助在命令层中对操作分块。

- 根据人们短期记忆的“7±2”或“每次记忆3块/每块3项”的特点，把菜单深度尽量限制在三层之内。
- 减少操作步骤：把点取、拖动和键盘操作减到最少。

4) 设计详细的交互

- ✓ 用户界面设计有若干原则，包括：
  - 一致性：采用一致的术语、一致的步骤和一致的活动。
  - 操作步骤少：减少敲键和鼠标点取的次数，减少完成某件事所需的下拉菜单的距离。

- 不要“哑播放”：每当用户等待系统完成一个活动时，要给出一些反馈信息。
- Undo：在操作出现错误时，要恢复或部分恢复原来的状态。
- 减少人脑的记忆负担：不应在一个窗口使用在另一个窗口中记忆或写下的信息；需要人按特定次序记忆的东西应组织得容易记忆。
- 学习的时间和效果：提供联机的帮助信息。
- 趣味性：尽量采取图形界面，符合人类习惯。

5) 继续做原型

- ✓ 开发用户界面原型，可对提交的人机交互活动进行体验、实地操作，并精炼成一致的模式。
- ✓ 使用快速原型工具或应用构造器，对各种命令方式，如菜单、弹出、填充以及快捷命令，做出原型让用户使用，通过用户反馈、修改、演示的迭代，使界面越来越有效。

6) 设计人机交互类

- ✓ 对窗口进一步细化，通常包括：类窗口、条件窗口、检查窗口、文档窗口、画图窗口、过滤器窗口、模型控制窗口、运行策略窗口、模板窗口等。
- ✓ 设计人机交互类，首先从组织窗口和部件的用户界面界面的设计开始。每个类包括窗口的菜单条、下拉菜单、弹出菜单的定义。还要定义用于创建菜单、加亮选择项、调用相应响应的操作。

7) 根据图形用户界面进行设计

- ✓ 建立图形窗口，确定字型、坐标系统。
- ✓ 建立事件响应机制。

- 设计人机交互子系统的启发式准则

❶每一个组织和用户都有其文化背景。可能不仅仅意味着语言、传统和习惯。由于所建立的系统面对的是用户，因此，其界面必须必须与用户的文化背景相一致。

一种适应用户文化背景的有效方法是“可视化表示”。目的是让计算机界面适应用户。例如有一个客户开发了一个财务管理软件，它以政府规定的各种计算方法和表格作为其可视化表示。对于这样的用户界面，学习和掌握它非常简单和容易。

❷ 使用用户开发的场景或使用事例来驱动界面。为避免用户界面太复杂，先观察用户是如何完成其工作的。在执行一个特定的工作时，用户界面应能告诉用户下面将做什么。

❸ 应当首先定义一个高层的用户界面和一些详细的对话框，然后定义人机交互对象，从而完成设计。需要建立原型对所有人机交互界面设计进行严格的检验。

❹ 人机交互子系统的设计应从建立分析模型时就开始着手。在开发系统的人机交互子系统时，应允许用户对其试用。

❺ 多数用户都不会从头开始设计人机交互类。事实上，使用各种所谓的可视化开发环境，如Delphi, PowerBuilder、Vasual Basic、Vasual C++、Borland C++ builder等，开发人员可能连人机交互子系统都不要。直接使用这些工具提供的控件，就可以作出用户界面。

用户可以不需要HIC，但不能免去用户界面的设计。使用菜单树或状态，连同某些原型，来说明用户界面的设计思想。

11.2.6 任务管理子系统

- 在应用中，每一个对象中的每一个服务最终都要被分配给某一个计算机任务。这样一些任务可以被看作是一些独立的可调度的实体。
- 任务管理子系统的任务为：
  - ✓将子系统映射到构件和处理器上
  - ✓标识并存储持久性数据
  - ✓提供访问控制
  - ✓设计全局控制流
- 通常利用UML构件图和部署图展示。

(1) 将子系统映射到构件和处理器

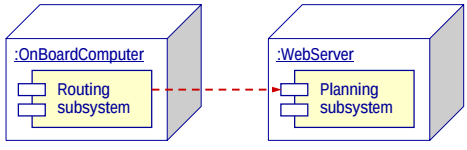
❶ 选择硬件配置和平台

✓许多系统运行在网络上。使用多台计算机，可以满足高性能计算需求和多个互联的分布式用户需求。确保将子系统分布到多台计算机上，设计基础设施来支持子系统之间的通信。

✓选择硬件配置包括选择虚拟机，系统将在其上运行。虚拟机包括操作系统和所需软件构件，如数据库管理系统和通信包。构件提供的功能越多，所需的开发工作量越少。

✓虚拟机的选择还受客户约束和成本限制影响。

✓ 例如，在车辆路程规划系统中，规划子系统和路由子系统运行在两个不同的节点上。硬件分配的情况如下：



将路由子系统分配到车载计算机上　　规划子系统分配到基于Web的网络服务器上

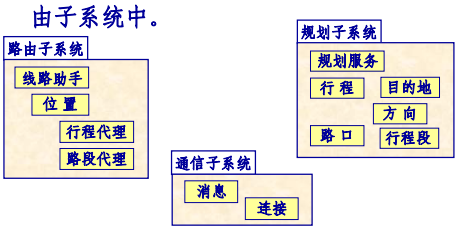
❷ 将对象和子系统分配到节点上

✓将对象和功能分配到各个节点上时，为了在节点之间传输数据，需对新加入的对象和子系统进行标识。

✓例如，在车辆路程规划系统中，路由子系统和规划子系统共享了行程类、目的地类、交差口类、路段类和方向类的实例，它们需采用一些通信协议，可通过无线调制解调器进行通信，可创建一个通信子系统来支持该通信。

✓旅途规划的内容分布在路由和规划子系统中，在路由子系统中创建路段代理类和行程代理类为规划子系统中的路段和行程提供代理服务。

✓ 当司机需要重新规划行程路线时，行程类对象会向通信子系统发出请求，检索规划子系统中有关路段的信息。最后，通信子系统将一个完整的旅途线路从规划子系统转移到路由子系统中。

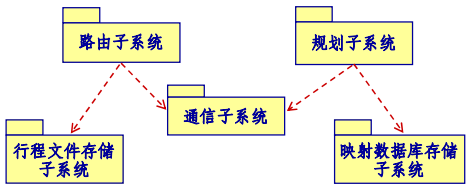


✓ 将某一子系统分配到硬件节点上，意味着能够将功能和处理能力分配到最需要这些子系统的地方。随之而来的问题就是子系统中的数据存储、数据转移、数据复制和同步数据等问题。

（2）标识并存储持久性数据

- 持久性数据的生存周期要长于系统的一次执行周期。例如，在图书管理系统中将借阅者借阅的图书信息和借阅者信息记入借阅记录并存储在数据库相应文件中，该文件以后可以再次打开。
- 数据在系统中的存储位置和存储方式将会影响到系统的分解。特定数据库管理系统的选择也会影响全局控制策略和并发管理。

- 例如，在车辆路程规划系统中，把当前行程信息存储在磁盘文件中是最简单、最有效的。为此，在系统中添加行程文件存储子系统和映射数据库存储子系统。
- 前者负责将旅程路线信息存储到车载计算机的文件中。该文件只支持整个旅程数据的快速存储和载入，以在车辆故障恢复时使用。后者负责将地图和旅程数据存储到规划子系统的数据库中。该子系统支持多个并发的的司机和规划代理。



❶ 标识持久性对象。候选的持久性数据是从分析过程中标识出的实体类对象，以及边界类对象。通常可以在系统关闭后，检查所有必须保存的类，以标识出必须长久保存的持久性对象。这里的系统关闭可以是受控关闭，也可以是系统崩溃。

❷ 选择存储管理策略。决定如何存储这些持久性对象的决策常常受到如下非功能属性的制约：

✓ 对象能否快速检索出来
✓ 是否必须执行复杂的查询来检索这些对象
✓ 这些对象是否需要大量的内存和磁盘空间

（3）提供访问控制

- 在多用户系统中，不同参与者对不同的功能和数据可有不同的访问权限。
- 例如，一个普通用户参与者可能仅能访问其所创建的数据，而一个系统管理员参与者对系统数据和所有用户数据具有无限的访问权限。

- 在需求获取和分析建模过程中，将不同用例关联到不同的参与者；在系统设计过程中，定义共享对象、参与者通过访问控制进行访问的权限、数据加密的方式等。
  - ✓例如，在车辆行程规划系统中，在同一数据库中存储地图信息和行程信息会引发安全问题，必须确保行程仅被发送给创建这些行程的司机。
  - ✓为此，可将司机类与行程类关联起来。规划子系统负责在发送行程之前对司机进行识别，最后根据两子系统之间的通信信息是否加密进行决策。

- 可以用访问矩阵对类的访问控制进行建模:
  - ✓矩阵的行列出系统中的参与者
  - ✓矩阵的列列出要进行控制的类
  - ✓矩阵元素列出参与者在类实例上能执行的一组操作，也称为访问权。
- 访问矩阵定义了静态访问控制。还可以采用代理模式定义动态访问控制。
- 例如，可以为每一个被访问的类实例创建一个代理对象，由代理对象检查访问者的访问权限。如果访问者具有相应的访问权限，则访问可执行，否则，访问失败。

(4) 全局控制流设计

- 典型的控制流机制有三种:
  - ① 过程驱动控制
  - ② 事件驱动控制
  - ③ 线程机制
- 一旦选择了控制流机制，就可以采用一个或多个控制对象来实现它。控制对象的作用是记录外部事件，存储其有关状态，给出接口上的操作调用、与外部事件相关联的实体对象操作调用的正确顺序。

11.2.7 数据管理子系统

- 数据管理子系统提供了在数据管理系统中存储和检索对象的基本结构，包括对持久性数据的访问和管理。
- 它分离了数据管理机制所关心的事项，包括文件、关系型数据库管理系统或面向对象数据库管理系统等。
- 数据管理的方法主要有 3 种: 文件管理、关系数据库管理和面向对象数据库管理。
- 下面给出选择数据管理方法的规则。

类型	选 择 规 则
文件	(1) 存在大量的数据（如图像） (2) 存在临时数据（如内存文件） (3) 存在低密度信息（如档案文件、历史日志）
关系型或面向对象数据库	(1) 存在并发访问 (2) 访问比较合适的细节层 (3) 对相同数据的多重平台或应用
关系型数据库	(1) 属性上的复杂查询 (2) 存在大数据量
面向对象数据库	(1) 为检索数据扩展关联的使用 (2) 中等规模的数据集 (3) 对象之间的不规则关联

- 文件是一种由操作系统提供的数据组织，应用程序以字符流的形式存储其数据。应用程序可定义用什么方式、在什么时间来检索这些对象。但文件使用时必须考虑很多问题，如并发访问和系统崩溃情况下的数据丢失问题。
- 关系数据库管理系统可以遵照预先定义好的结构类型进行存储。它使用若干表格来管理数据，表中每一列代表一个属性，每一行代表一个属性元组值的数据元素。这种类型的数据库提供了并发管理、访问控制和故障恢复服务，但对非结构化数据（如图像、自然语言文本），速度较慢。

- 面向对象数据库管理系统提供的服务与关系型数据库管理系统类似。不同之处在于，面向对象数据库以对象和关联的方式存储数据。它还提供了继承和抽象数据类型，极大减少了存储子系统的开发时间，但其查询速度慢，且不容易调试。

11.3 对象设计

- 对象设计的主要任务是追加解空间的对象和对已有对象进行细化。对象设计包括：
  - ①复用：寻找可利用的已有解决方案，包括可复用类库、商业外购构件和设计模式。
  - ②服务规格说明：精确描述每个类的接口。
  - ③重构对象模型：改进对象设计模型，以提高可读性和扩展性。
  - ④优化对象模型：改进对象设计模型，以满足性能标准。

11.3.1 使用模式设计对象

- 对象设计的一个任务是在应用对象与系统设计所标识的硬件／软件平台之间建立连接。
- 开发人员可以复用已有的类，建立设计模型中已定义解对象的细化对象，或追加一些自定义的解对象。还可以复用设计模式。复用的 4 个步骤为：选择、分解、配置和演变。

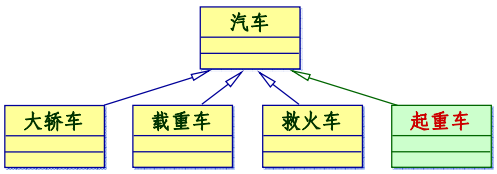
1. 选择
   - ✓ 开发一个新对象最简单的方法是从已有构件中简单地选择合乎需要的软件构件。

- ✓为此，需要利用一些商业外购构件库或遗留软件构件库。这些构件库提供以下层次的构件：
  - ①特定小组的可复用构件（一个小组为他们自己组内所有成员使用而开发）
  - ②特定项目的可复用构件（一个小组为某一个项目而开发）
  - ③特定问题领域的可复用构件（购自某一个特定领域的软件销售商）
  - ④通用构件（购自专门提供构件的销售商）
  - ⑤特定语言原操作（购自一个编译器的销售商）

2. 分解
   - ✓ 最初标识的“类”常常是几个概念的组合。在着手对象细化设计时，可以把一个类分成几个类，使得新标识的类容易实现，或它们可以从已有的类中获得。
3. 配置
   - ✓ 在设计类时，可能会要求由已有类的实例提供类的某些特性。通过把相应类的实例声明为新类的属性来配置新类。
   - ✓ 例如，一种仿真服务器可能要求使用一个计时

器来跟踪服务时间。设计者应找到计时器类，并在服务器类的定义中声明它。这个服务器还要求有一个队列类的实例来作客户排队工作。

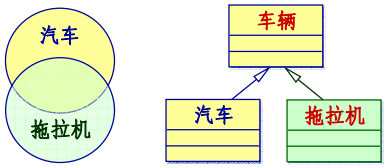
4. 演化
   - ✓ 要创建的新类可能与一个已有类非常类似，但不完全相同。此时，可以利用继承机制。调整继承结构，加入合乎需要的新类的处理有三种可能的方式。



a) 建立一个新类起重车。它的底盘、发动机、轮胎、驱动装置等都在已有类汽车中。关系如右图所示。新类是已有类的特殊情形。这时直接让起重车类作为汽车类的子类即可。

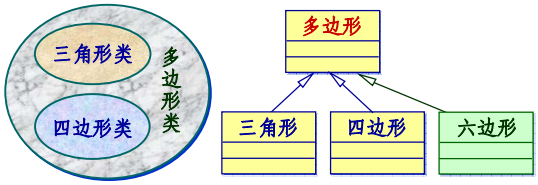
b) 增加一个新类拖拉机。它的底盘、发动机等与汽车不同，但驱动装置、轮胎等与汽车相同。

- ✓ 调整继承结构。建立一个新的一般的车辆类，把拖拉机与汽车类的共性放到车辆类中，拖拉机与汽车类都成为车辆类的子类。
- ✓ 车辆是抽象类，相关操作到子类汽车找。



c) 另一种情形是在已有类的基础上加入新类，使得新类成为已有类的泛化类。

- ✓ 例如，已经存在三角形类，四边形类，想加入一个多边形类，并使之成为三角形和四边形类的泛化类。



11.3.2 设计接口规格说明

- 接口规格说明的目标：足够清晰地描述每一个对象的接口，为实现这个对象打下基础。设计接口规格说明的主要活动包括：

1. 标识遗漏的属性和操作
   - ✓ 检查每一个子系统提供的服务以及每一个分析对象；标识需要子系统服务实现，但却遗漏的属性和操作。为此对当前对象设计模型细化，对所用到的操作参数细化。

2. 标识类型、可见性和签名
   - ✓ 确定哪些操作可以提供给外部其他对象或子系统使用，哪些操作只能属于同一类的对象。还要说明每一个操作的返回类型、参数的数目和类型。
   - ✓ 本活动的目标是建立并提供一个较小且简单的接口，便于使用者理解和使用；同时降低子系统之间的耦合度，便于系统的更新。
3. 描述前置条件和后置条件
   - ✓ 为每个类的所有公有操作订立契约，即存在于类使用者和类实现者之间的一个协议。通过前

置条件描述类使用者必须遵守的那部分契约，即操作正确执行必须满足的先决条件；后置条件描述类使用者在遵守相关部分契约之后，类实现者应当保证相关操作能够正确执行，并在执行之后契约得到满足。

3. 描述不变式
   - ✓ 不变式是一种与类和接口有关的约束，通常用来表明类的实例中所有属性和操作都必须遵守的一种契约。
   - ✓ 例如在保龄球赛类中有 3 个方法：addPlayer() 增加一个选手、removePlayer() 删除一个选手

和getMaxPlayers() 求参赛选手的最大数目。

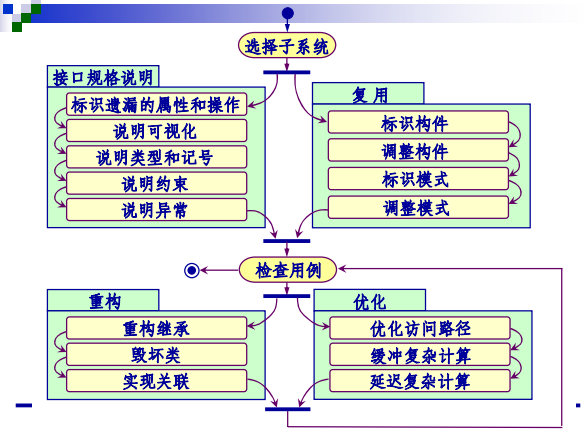
- ✓ 在构造一个保龄球赛对象 B 时，若不变式为 B.getMaxPlayers() = 0，则使用 B.addPlayer() 将违反该操作的契约。因此应将不变式改设为 B.getMaxPlayers() > 0。
- ✓ 编写不变式的目的是为了说明类实现者为类使用者所做的约束。因此，类实现者应注意这种约束应是简单的、明确的边界情况。

11.3.3 重构对象设计模型

- 重构的目的是通过软件复用来增加设计模型中的某些不足部分，或满足其他的设计目标。
- 典型的重构活动的例子包括:
  - ✓将一个 N 元关联转换成一组二元关联;
  - ✓将两个不同子系统中相似的类合并为一个通用的类;
  - ✓将没有明显活动特征的类转换为属性;
  - ✓将复杂类分解为几个相互关联的简单类;
  - ✓重新组合类和操作，增加封装性和继承性;

11.3.4 优化对象设计模型

- 优化的目的是实现设计模型中提出的性能要求。
- 典型的优化活动的例子包括:
  - ✓ 选择更高效的算法来提高系统执行的速度;
  - ✓ 使用更大、更高效的存储系统;
  - ✓ 减少连接中的重复性来提高查询的速度;
  - ✓ 为了提高效率，增加额外的连接;
  - ✓ 改变执行的顺序;
  - ✓ 增加导出属性，减少对象操作的执行时间等。
- 对象设计的过程的各项活动是并发执行的。



- 先进行接口规格说明和复用活动，产生一个对象设计模型。
- 再使用这个模型检查某个特定子系统的用例，即该子系统的功能。
- 一旦该子系统的对象设计模型稳定下来（即不再需要大的修改）后，再进行重构和优化。如果在设计接口、构件和设计模式等方面投入较多，就可以产生一个易于修改的对象设计模型。
- 对象设计过程是一个迭代的过程。接口规格说明、复用、重构、优化等活动需要进行多次反复设计。