

## 第八章 面向对象的系统开发方法

### 内容提要

本章围绕面向对象技术的基本概念和运行机制，结合应用实例，介绍面向对象的系统开发方法。

### 第一节 面向对象的基本概念及特征

#### 内容提要

- 面向对象有关概念
- 面向对象的基本特征

#### 学习目的

- 了解面向对象的思想
- 掌握对象、类、消息、继承等基本概念
- 了解面向对象的特征

#### 一. 面向对象的基本概念

##### 1. 面向对象

面向对象是一种认识客观世界的世界观，是从结构组织角度模拟客观世界的一种方法，人们在认识和理解现实世界的过程中，普遍运用以下三个构造法则：

- (1) 区分对象及其属性，如区分车和车的大小；
- (2) 区分整体对象及其组成部分，如区分车和车轮；
- (3) 不同对象类的形成及区分，如所有车的类和所有船的类。

##### 2. 对象（Object）

对象是对一组信息及其上同的操作的描述。如：

一辆汽车是一个对象，它包含了汽车的信息（如颜色、型号、载重量等）及其操作（如启动、刹车等）；

一个窗口是一个对象，它包含了窗口的信息（如大小、颜色、位置等）及其操作（如打开、关闭等）。

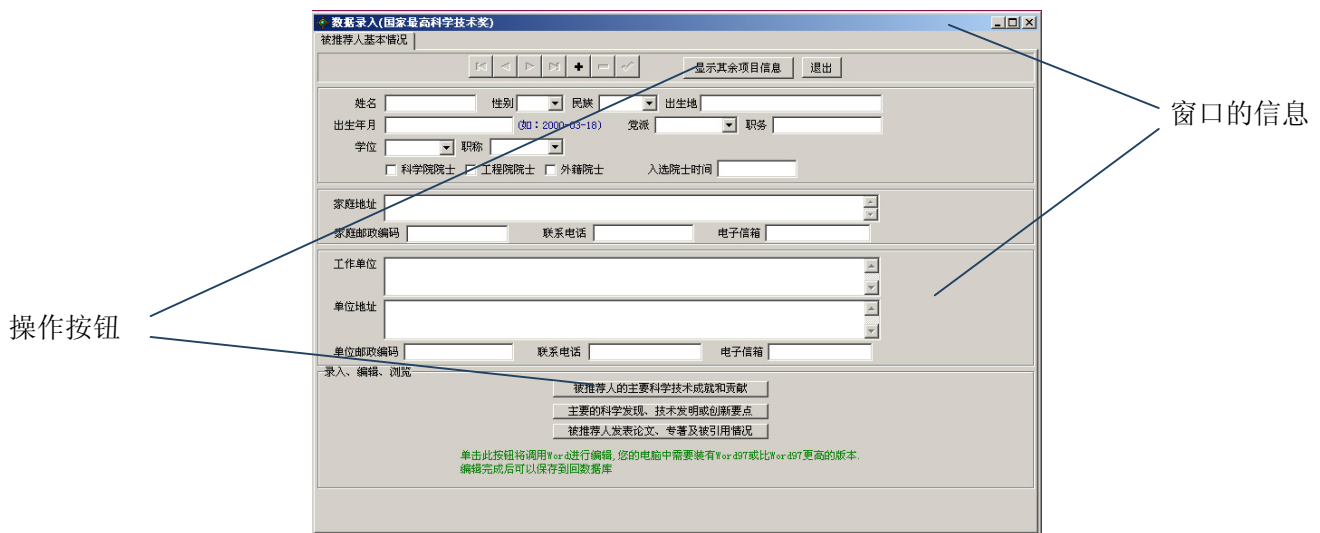


图 8.1 窗口对象

### 3. 属性 (Property)

即对象所包含的信息。

如：窗口的宽度（`form1.width`）

属性可以在设计对象时确定，也可以在程序运行时读取和修改（如：`W=form1.width; form1.width=100`）。

### 4. 方法(Methord)

即对象所因有的各种操作。

如：窗口关闭（`form1.close`）

这种操作的过程对外是封闭的，即用户只能看到这一方法实施后的结果。这相当于事先已经设计好的各种过程，只需要调用就可以了，用户不必去关心这一过程是如何编写的，事实上，这个过程已经封装在对象中，用户也看不到。

对象的这一特性，即是对象的封装性。

### 5. 事件 (Event)

即对象在执行某一操作后激发并执行的一个或多个过程。

这些过程对用户是透明的，用户可以为这个过程编写自己的程序代码，以完成特定的操作。

如：窗口对象在执行打开过程时，就会激活一个 **Active** 事件（过程），用户可以自己编写这一过程的代码，以便在打开这个窗口时完成一些自己所要求的任务，如打开一个数据库，对某个变量进行初始化等等。

### 6. 类 (Class) 与实例 (Instance)

类是具有共同属性、共同方法、共同事件的对象的集合。而一个具体的对象则是其对应类的一个实例。

如果对某一个类的定义进行修改,如增加一些属性或修改一些方法,就得到一个新的类,而原先的类就是新类的父类。

如：我们在窗口类的定义上，另外定义它还包含两个按钮，一个标题是“确定”，一个标题是“取消”，并且在分别按下这两个按钮后，执行“OK”与“Cancel”两个事件。这样就定义了一个新类，我们可以把它叫做“选择窗口”类，并保存它，这样就可以直接由“选

择窗口”生成窗口实例，而不用每次都由“窗口”类产生，然后再添加按钮。

## 7. 继承 (Inheritance)

任何一个子类都具有其父类所有的属性、方法、事件。这一特性叫做类的继承。

如果父类的特性发生变化，其子类也相应改变。

继承机制的优点在于：

- (1) 避免了由于系统内类对象封闭而造成数据和操作冗余的现象。

每个子类都可以继承其父类的特性，包括状态与行为。同时子类可以有与父类不同的地方，即子类可根据自身特点新增或局部修改(modification)父类的行为而加以使用，甚至可以覆盖父类中的定义。利用继承，我们只要在原有关的基础上修改、增补、删减少量的数据和方法，就可以得到子类，然后生成大小、初态不同的实例。

(2) 接口的一致性(Consistency of interface)，父类衍生子类的其他操作接口也传递给其子类。

- (3) 符合软件可重用性。

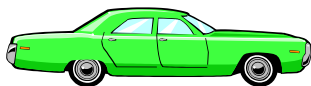
传统的结构化方法中的过程调用，以及类定义出对象，都是重复使用的典型例子。但它们都比不上继承的层次高。因为通过继承明显使软件开发速度加快，实现较高级别的共享。这是继承最重要的优势。

## 8. 消息(message)

面向对象的世界是通过对象与对象间彼此的相互合作来推动的，对象间的这种相互合作需要一个机构协助进行，这样的机构可以称为"消息传递"。消息传递过程中，由发送消息的对象(SENDER)的使动操作产生输出结果，做为消息 (MESSAGE) 传送至接受消息的对象 (RECEIVER)，引发接受消息的对象一系列的操作。所传送的消息实质上是接受对象 (RECEIVER) 所具有的操作/方法名称，有时还包括相应参数，图 8.2 就表示了这样的概念。



图 8.2 消息传递模型



行驶    时速 50 公里

说明：

点击车下面的“行驶/时速 50 公里”的按钮，车即开始移动。(动画)

注意，“消息传递”机制与传统的“过程调用”机制的意义是截然不同的，发送消息仅

仅是触发对象自动机，有时提供一些附加数据，接受对象响应消息后按照消息模式找到匹配方法，因而同样的输入参数(触发事件)可能因对象状态的不同得到不同的终态(若有输出则结果不同)。而过程调用则相反，只要有相同的输入，输出总是恒定的。但不否认可以借用“过程调用”机制来实现“消息传递”，此时只要在匹配方法后考虑到接受对象的当前状态即可。因而，系统可以简单地看作一个彼此通过传递消息而相互作用的对象集合。

总之，面向对象的整体概念可具体表示如下：

面向对象 = 数据抽象 + 数据抽象类型 + 继承性

## 二. 面向对象的特征

面向对象具有以下特征：

### 1. 封装性

对象的概念突破了传统数据与操作分离的模式。对象作为独立存在的实体，将自由数据和操作封闭在一起，使自身的状态、行为局部化。

### 2. 继承性

继承是面向对象特有的，亦是最有力的机制。通过类继承可以弥补由封装对象而带来的诸如数据或操作冗余的问题，通过继承支持重用，实现软件资源共享、演化以及增强扩充。

### 3. 多态性

同一消息发送至不同类或对象可引起不同的操作，使软件开发设计更便利，编码更灵活。

### 4. 易维护性

面向对象的抽象封装使对象信息隐藏在局部。当对象进行修改，或对象自身产生错误的时候，由此带来的影响仅仅局限在对象内部而不会波及其他对象乃至整个系统环境，这极大方便了软件设计、构造和运行过程中的检错、修改。

## 第二节 面向对象的系统开发方法的原理

### 内容提要

- 面向对象系统开发的基本特征
- 面向对象系统开发的阶段划分
- 面向对象系统开发的思路

### 学习目的

- 了解面向对象系统开发的特征
- 了解面向对象系统开发各阶段的开发思路

## 一. 面向对象开发方法的内容与过程

面向对象开发一般经历三个阶段：面向对象系统分析(OOA)，面向对象系统设计(OOD)

和面向对象系统实现(编程)。这与传统的生命周期法相似，但各阶段所解决的问题和采用的描述方法却有极大区别。

图 8.3 表示的是面向对象系统开发模型，它表达了面向对象开发的内容和过程。

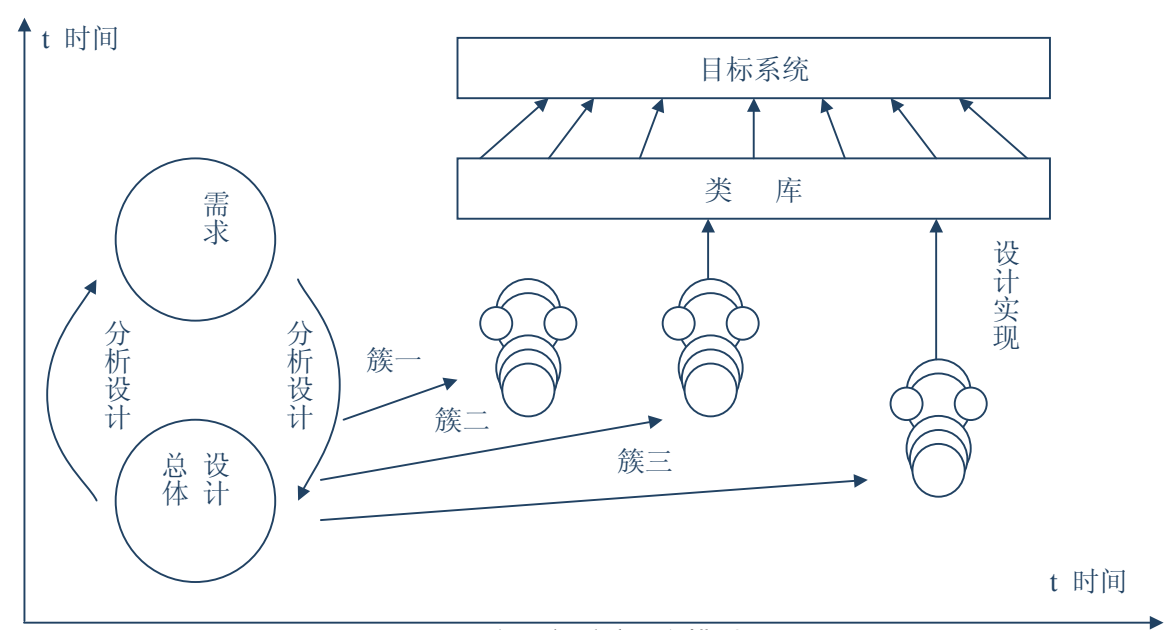


图 8.3 面向对象系统开发模型

语音：

系统人员通过需求调查，在反复的分析设计中不断地构建出“簇”，(cluster)。簇的概念是 B.Meyer 提出来的。他在研究面向对象开发过程中感觉到用单个对象映射客观实体难以实现，所以就用一组对象来为客观世界的复杂实体建模。所谓“簇”就是一组对象。一方面构建的簇经过设计实现被存入系统的类库中去以各再用；另一方面，构建“簇”的这组对象也可来源于系统已存在的类库。

## 二. 面向对象开发方法的基本特征

上述面向对象开发模型体现了面向对象开发方法的基本特征：

- (4) 分析与设计是反复的，充分体现了原型开发的思想；
- (5) 分析与设计的不断反复结果是对客观世界对象的模型化，建立针对簇的规格说明；
- (6) 运用库中已有对象，反复测试实现簇，并将新簇纳入库中，这一过程体现了继承和重用；
- (7) 强调分析阶段和设计阶段的合并。

### 面向对象开发方法各阶段的思路

#### 1. 分析阶段

这一阶段主要采用面向对象技术进行需求分析。面向对象分析运用以下主要原则：

- (1) 构造和分解相结合的原则。构造是指由基本对象组装成复杂或活动对象的过程；分解是对大粒度对象进行细化，从而完成系统模型细化的过程。
- (2) 抽象和具体结合的原则。抽象是指强调事务本质属性而忽略非本质细节；具体

则是对必要的细节加以刻划的过程。OO 方法中，抽象包括数据抽象和过程抽象：数据抽象把一组数据及有关的操作封装起来，过程抽象则定义了对象间的相互作用。

(3) 封装的原则。封装是指对象的各种独立外部特性与内部实现相分离，从而减少了程序间的相互依赖，有助于提高程序的可重用性。

(4) 继承的原则。继承是指直接获取父类已有的性质和特征而不必再重复定义。这样，在系统开发中只须一次性说明各对象的共有属性和[服务](#)，对子类的对象只须定义其特有的属性和方法。继承的目的也是为了提高程序的可重用性。

**服务：**指的是对象收到消息后所执行的操作。

二是构造问题空间。面向对象方法构造问题空间时使用了人们认识问题的常用方法，即：

- (1) 区分对象及其属性，例如区分一棵树和树的大小或位置；
- (2) 区分整体对象及其组成部分，例如区分一棵树和树枝，在面向对象方法中把这一构造过程称为构造分类结构；
- (3) 不同对象类的形成及区分，例如，所有树的类和所有石头的类的形成和区分。在面向对象方法中把这一构造过程称为组装结构。

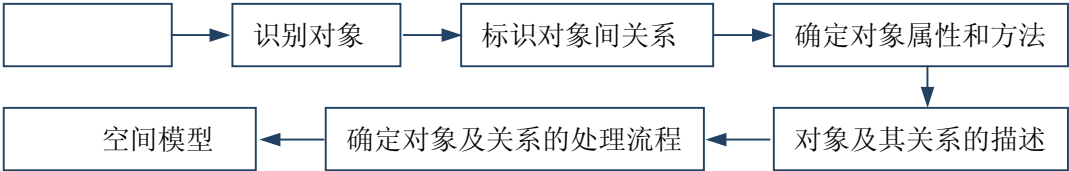


图 8.4 面向对象系统分析的过程

2. 设计阶段

这一阶段主要利用面向对象技术进行概念设计。值得注意的是面向对象的设计与面向对象的分析使用了相同的方法，这就使得从分析到设计的转变非常自然，甚至难以区分。可以说，从 OOA 到 OOD 是一个积累型的扩充模型的过程。这种扩充使得设计变得很简单，它是从增加属性、服务开始的一种增量递进式的扩充。这一过程与结构化开发方法那种从数据流程图到结构图所发生的剧变截然不同。

一般而言，在设计阶段就是将分析阶段的各层模型化的“问题空间”逐层扩展，得到下个模型化的特定的“实现空间”。有时还要在设计阶段考虑到硬件体系结构，软件体系结构，并采用各种手段(如规范化)控制因扩充而引起的数据冗余。

3. 实现(编码)阶段

这一阶段主要是将 OOD 中得到的模型利用程序设计实现。具体操作包括：选择程序设计语言编程、调试、试运行等等。前面两阶段得到的对象及其关系最终都必须由程序语言、数据库等技术实现，但由于在设计阶段对此有所侧重考虑，故系统实现不会受具体语言的制约，因而本阶段占整个开发周期的比重较小。

建议应尽可能采用面向对象程序设计语言，一方面由于面向对象技术日趋成熟，支持这种技术的语言已成为程序设计语言的主流；另一方面，选用面向对象语言能够更容易、安全和有效地利用面向对象机制，更好地实现 OOD 阶段所选的模型。

第三节 面向对象的系统分析和设计实例

内容提要

- 面向对象的系统分析和设计的过程
- 面向对象的系统分析和设计的内容
- 面向对象的系统分析和设计的方法

## 学习目的

- 掌握面向对象开发方法的基本步骤

## 一. 面向对象开发的常用方法

90 年代初，对利用面向对象技术进行系统开发的研究进入了百花齐放、百家争鸣的繁荣阶段，涌现出许多面向对象的系统开发方法及建模方法，其中已形成完整体系结构的有 Shlaer&Meller's, OOA&OOD 方法、Booch's OOA&OOD 方法、GOOD(General Object Oriented Design)方法、James Rumbaugh 的 OMT 方法、Wirs-Brock 方法和 Coad&Yourdon 的 OOA&OOD 方法。

## 二. 面向对象的分析与设计举例

下面我们以 Coad&Yourdon 的方法为基础，结合实际应用，详细地介绍面向对象系统的分析和设计。

### 一、面向对象的分析

#### (一)问题陈述

这里所举的例子是一个车辆注册管理系统。对该问题域的陈述如下：

车主在购入车辆后，执相关有效证件到主管部门，找到具体负责的工作人员进行登记注册，缴纳一定费用，获得相应牌照。注册后，有关车况信息和车主信息要备案。

系统所需维护的信息有：

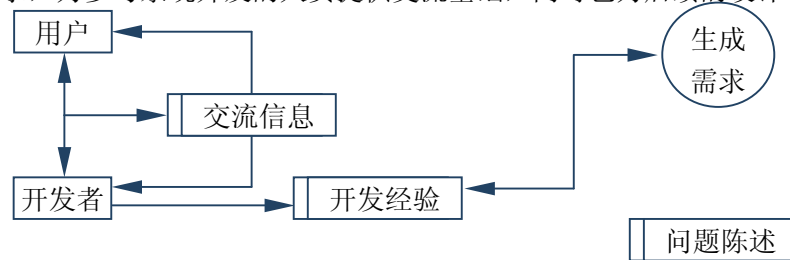
信息类型	信息内容
主管部门信息	名称、负责人、地址、电话传真等
具体工作人员信息	姓名、权限、工作年限等
车主信息	姓名、住址、联系电话等
登记信息	流水号、车号、所有权、凭据、放弃登记标识及费用等
注册发照信息	起始/终止时间、品牌(发动机出厂号，年份，种类，牌号)、标签(年份，品种，号码)、费用
车辆信息	车号、出厂日期、制造商、车型、总重、载容量、内燃机马力、颜色、价格、已行驶里程等

工作人员负责登记发牌照，收取费用。每个顾客都来自一个地区，属于某一部门。系统不保留有关牌照格式、车牌或标签号的清单。

值得注意的是上述对问题空间的理解可能是不完整的，亦可能是二义的(自然语言表达时不可避免的)，甚至可能是不一致的。这就需要在系统分析阶段明确、扩展、细化对问题的陈述。

#### (二)系统分析

这一阶段主要是根据已有的问题空间的描述,采用面向对象分析方法,为现实应用领域建立相应模型,整个分析过程如图 8.5 所示。分析过程得到的模型能明确地刻划出系统的需求,为参与系统开发的人员提供交流基础,同时也为后续的设计和实现提供基本框架。



### 1. 标识对象

**标识对象：**将现实应用中的实体与目标系统中的技术概念更加紧密地联系在一起，并构造一个稳定的框架作为应用领域模型的基础。

开发人员定义对象应首先从已得到的问题陈述入手,在此基础上反复对用户业务流程进行调查,研究用户提供的有关系统需求的形式不一的文字资料,查阅与应用领域紧密相关的专业文献,加强同用户进行及时的面对面的交流,研究所有尽可能得到的图示资料,包括系统组成图、高层数据流程图,从而获得对问题空间的深居的较完整的理解,并在此基础上尽量捕捉到与系统潜在对象相关的信息。下面列出有关准则:

(1) 搜寻准则。挖掘系统潜在对象时,要依次考虑以下几类事物:

① 结构: 主要考虑分类和组装两种结构,这不仅能发现对象,还可以明确系统层次关系;

② 其他系统: 是指与本系统相互作用的系统或"外部边界"。这种相互作用包括硬件链接,信息互传或实体相互作用。本例中"车辆"就属于与本系统实体相互作用的例子;

③ 设备: 指与系统作用的有关设施,有些可能与系统进行数据或控制信息的转换;

④ 需存贮的事件: 指问题域中发生的需要保存相关信息的事件,包括时间、地点、人物、原因等因素在内都需系统维护;

⑤ 人员作用: 系统中人员通常分两种: 其一是系统直接使用者,亦称用户; 其二是系统所处理信息的源主,在本例中即指"车主";

⑥ 地点: 指系统需考虑的物理地点、办公室或场所;

⑦ 组织单元: 指与系统有关的人所属的地域、部门或机构。



按搜寻准则对系统进行扫描，一旦发现候选对象，就参照判别准则来取舍，并利用检验准则做最终的审查。

(2) 判别准则。当决定模型中是否包含某一对象时，至少要考虑以下四点：

① 系统是否有必要保存该对象信息，为该对象提供服务

② 对象的属性至少大于 1。利用这条准则过滤掉低层次上的一些对象。

③ 公共属性及服务的确认。若确实是公共属性和服务，则抽象出来用以产生实例；否则需用分类结构进行说明。

④ 基本要求。即在不考虑具体实现系统的计算机技术时，系统必须有的需求。

(3) 检验准则。在经历了对问题空间的搜索找到对象，并对这些对象进行判别后，我们得到一些使用自然语言描述的候选对象，究竟这些候选对象是否符合要求，还要经过严格的检验。

① 冗余的属性和服务。若系统在时间、进度、能力三方面的制约下，不必存贮某些属性数据或提供某类服务，那么就删除这些属性数据及服务对应的对象。

② 单个实例对象。这条准则主要针对有属性的对象，分三种情况考虑：若单个实例对象确实反映问题空间中的实体，那么其存在是合理的；若系统中还存在另一个有相同属性和服务的对象，并且它也正确刻划问题域，则将二者合并；若系统中存在另一个有相似属性和服务的对象，且它也能正确刻划问题域，则考虑使用分类结构。比如本例中“登记”和“注册发照”两个事件对象就属于第三种情况，需要构造一个“合法事件”类。

③ 派生结果。模型中不能有派生结果，但模型中需要保存能够得到派生结果的对象。

在确定对象后，需要为对象命名，即将非形式化的描述转化为形式化的描述。一个对象名应该能够描述对象的单个实例，它通常是单个名词，或是形容词+名词，并且是能够反映对象主题的标准词汇，还要具有较强的可读性。

对应本例的问题陈述，我们可以得到本系统的六个对象(如图 8.6 所示)。

## 2. 标识结构

结构表示问题空间的复杂度，标识结构的目的是便于管理问题域模型的复杂性。在系统分析中我们需要考虑的结构有两种：分类结构和组装结构。

### (1) 分类结构

它能够帮助我们得到成员组织层次，它通过搜集问题域中的公共特性并把这种特性扩充到特例之中来，显示现实世界实体的通用性及专用性。分类结构还提供了对问题空间的重要划分：一种划分是把属性和服务分成互斥的几组，另一种划分是利用结构抽象出比对象和结构都要高的数据层次，即“主题”。

图 8.7 所示是本例中的一个分类结构，“客车”、“卡车”等对象通过这个分类结构共享有关“车辆”的公共属性。“车辆”的属性是通用的，同时，“客车”等对象还可根据自身特点，增加属性(如“A”旁所示)。增加的属性不能为其他对象所用，是专有的。服务的共享及扩充原理亦如此。值得注意的是，公用属性和服务在结构中仅出现一次。

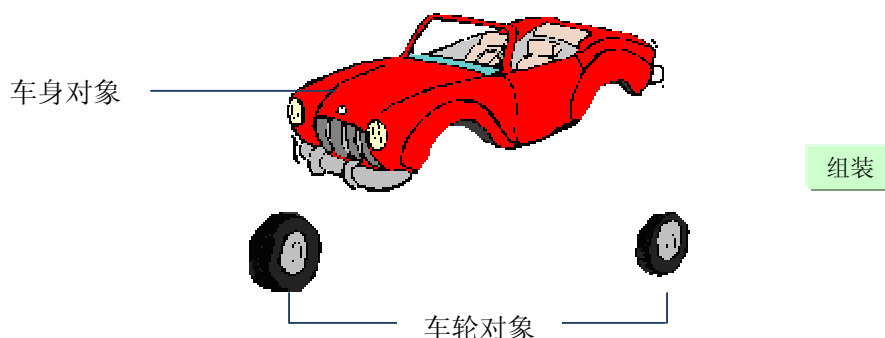
定义分类结构需先将一个对象考虑成通用的，并考察它在应用领域各种可能的专用特性。例如，本例中的对象“车辆”可按不同的专用性分为以下几类：商用和私人；载货和载客；汽油车和柴油车；轻型车和重型车。

接下来要考虑各专用性之间是否存在差异，明确某专用性的确存在于现实世界。此外，还要考虑这种专用性是否存在于问题空间，例如，要考虑是否把“车辆”分为载货和载客。这时，要把对象作为专用的来考察，考虑系统是否有其他对象通用，这种通用性是否反映现实世界，是否在系统范围内。在本例中，我们分别看到“客车”、“卡车”、“摩托车”等对象，就可以将它们综合成对象“车辆”。

图 8.8 是本例子的两个分类结构：“合法事件”及“车辆”。

## (2) 组装结构

组装结构刻画了一个整体及组成部分，表达了一种基本组织方式，即部分聚合成整体的方式。



说明：

当点击“组装”按钮时，由车身和车轮组成汽车。(动画)

接下来，我们要将每一个对象当作一个部分来考虑，考察该对象是否适用于组装，它与哪些对象在一起形成一个组装，以及该组装是否反映现实世界的实体，是否属于问题空间。本例的组装结构如图 8.9 所示。

由图 8.9 可以看到，组装结构的增加是通过从整体到部分，从顶到底的描绘而成的。用图中的一个短竖代表一个单独部分，用爪形标识表示多重部分。当一个部分仅可出现在一个组装中时，则在靠近组装结构处再标识一个短竖，对象层图中添入上述三个结构即得到结构层图示。

## 3. 标识主题

主题 (subject) 提供给开发人员一种控制机制，以把握在某个时间内所能考虑并理解的模型规模，并便于了解模型的概貌。采用主题机制还可获得方便的通讯能力，避免参与开发人员之间的信息过载，弥补对象、结构机制不能反映系统模型整体构成、动态变化以及功能信息的不足。

定义主题分为二步：

(1) 选择主题。需要先给每个结构标志一个相应主题，给每个对象标志一个相应主题，再考虑主题数目。如果主题的个数超过 7 个，则需进一步提炼主题。

(2) 构造主题层。列出主题及主题层上各主题间的消息连接(用箭头表示)，对主题进行编号，画一个简单的矩形框并配以合适的名字来表示一个主题。本例中的主题层如图 8, 10 所示。

## 4. 定义属性

定义属性是分析与选择的过程，大致要经过四个步骤：

(1) 标识、定位属性。标识属性首先要明确某个属性究竟在描述哪个对象，要坚持保证最大稳定性和模型一致性；其次要坚持在原子概念的层次上标识属性，比如，驾驶员的驾照号码可以是数据单元的组合，从而相应减少属性名。至于消除数据冗余的规范化问题，则要到设计阶段再做相应考虑。

属性标识完成后，要利用继承机制给属性定位：将通用属性放在结构的高层，将特殊的属性放在结构低层；若一个属性适用于大多数的特殊分类，可将其放在通用的地方，然后在

不需要的地方把它覆盖(override); 如果发现某个属性的值有时有意义, 有时却不适用, 则应考虑分类结构。

(2) 标识[实例连接](#)。

**实例连接:** 指一个实例与另一个实例的映射关系。

标识实例连接分三步完成:

① 添加实例连接线。将系统中必须维持的实例间的对应关系用连线表示, 每一条实例连线都意味着有一条相对应的消息连接线。当每个隐含连接标识对修改时, 一端实例就需要向另一端的实例发送一条消息。比如, 在本例中如果一个“车主”与“车辆”之间总有一个“合法事件”的实例发生, 那么模型中就隐含了“车主”和“车辆”之间的连接。

② 定义多重性和参与性。先对实例连接的每个方向考察其多重性: 一对一(1: 1), 还是一对多(1: M), 亦或是多对多(M: M)。本例中: “车主”与“合法事件”是多对多(M: M)的关系。

接下来要定义参与性, 明确在连接的两个方向上, 对象间的实例连接是强制性的还是任意性的, 即连接是否必须存在。如图 8.12 所示“车主”、“合法事件”及“车辆”之间的连接是必须的, 标注“1”; 而“具体工作人员”与“合法事件”的连接具有任意性, 标注“0”, 可以理解为对于一个新职员, 可能还要经过一段时间才允许处理正式的法定事件。

③ 检查特殊情况。包括三个以上对象或分类结构之间的连接, 多对多的实例连接, 相同对象或分类结构之间的实例连接, 及两个对象或分类结构之间的多重实例连接等几种情况。检验多对多的实例连接, 实质是检验对象间的连接中是否存在描述对象的属性。图 8.13 即为本例中的一个多对多的实例连接及相应措施(扩充标识)。

(3) 修订对象。随着属性的增加, 需要重新修订对象或分类结构, 主要有以下几个检验点:

① 带有“非法”值的属性。主要指只适合某些特定的实例的属性, 可引入附加的分类结构予以解决。

② 单个属性。单个属性作为对象易引发模型膨胀。若某个对象只有一个属性, 则修订模型, 将单个属性直接放入相关对象, 并删除多余的对象, 如图 8.14 所示。

③ 属性值冗余。若存在重复的属性值, 则考虑新增对象。但该新增对象必须符合对象标识准则, 而且要检查对象属性个数是否大于 1。

④ 适应性参数。属性值由操作决定, 在一定范围内选取, 处理方法是将每个的属范围或限制本身作为一个属性。这种方法的局限在于增加模型和对象中的属性个数。

(4) 说明属性和实例连接约束。用名字和描述性语言说明属性, 同时还可以增加一定的属性约束(取值范围、限制、计量单位和精度), 并且要将属性划分成以下几类:

- ① 描述性的(descriptive) — 其值在实例增加、修改、删除和选择时建立及维护;
- ② 定义性的(definition) — 其值可能适用于某个对象或分类结构的多个实例;
- ③ 可推导的(always derivable) — 其值在任何时候都由其他数据推出(不必保存);
- ④ 偶尔可推导的(occasion derivable) — 其值偶尔可推导(必须保存)。

接下来要对实例连接约束进行说明, 主要通过观察实例间的映射限制得到说明, 其中对组装连接的关系约束的说明也包括在内。

下面给出本例中的“合法事件”分类结构的属性及实例连接约束的说明:

① 描述性属性

合法事件

日期: 合法处理发生的日期和时间

登记

流水号: 合法处理的主记录号

所有权证明：证明所有权的文件证据  
放弃登记：放弃登记的理由及放弃的登记号  
注册发照  
起始时间：申请的开始日期及时间  
终止时间：申请的结束日期及时间  
牌照：出厂牌、年份、型号、号码  
标签：标签上年份、型号、号码

② 可推导属性

合法事件

登记、缴费：登记时交费

注册、缴费：注册发照时收费

③ 实例连接约束

与“车主”1：M 必须的

与“车辆”1：1 必须的

本例属性层示意图见图 8.15。

## 5. 定义服务

服务是指对象收到一条消息之后所执行的处理。定义服务的核心就是为每个对象和分类结构定义所需要的行为，下面就是三种常见的行为以及相应标识服务的策略：

(1) 有直接动因的行为：直接动因 — 状态 — 事件 — 响应。

(2) 进化史上的相似行为：进化史 — 对象生命历程。

(3) 功能相似的行为：功能 — 最基本的服务。

定义服务还必须解决的问题是确定对象实例之间必要的通讯。定义服务的策略分四步：标识服务、追加服务、标识消息连接、详细说明服务。

(1) 标识服务(亦称基本策略)。针对系统内每一个对象或分类结构考虑三类基本服务：Occur（实例增、删、改），Calculate（运算），Monitor（监测）。所有模型都使用 Occur 服务；部分模型中，当一个实例需要另一个实例说明的处理结果时，使用 Calculate 服务；模型中的实际处理部分使用 Monitor 服务。

(2) 追加服务(亦称辅助策略)。首先需要定义基本的对象历程序列，然后扩展每一步反映出增加、修改、删除和选择的演变。考虑对象或分类结构是否还响应其他事件，如果是则增加基本序列，根据需要增加基本服务，再利用状态 — 事件 — 响应这个策略发现对象或分类结构的附加服务。首先定义系统主要状态，然后对应每一个状态列出其外部事件和所需要的响应，建立一个状态 — 事件 — 响应表，最后为响应每个状态提供必要的服务处理。辅助策略有助于发现易被遗漏的 Calculate 类和 Monitor 类服务。

(3) 标识消息连接。消息连接结合了事件响应和数据流两个方面。在标志消息连接时，首先在已存在实例连接的对象和分类结构间增加消息连接，然后检查对象和分类结构(包括封装在其中的属性)，寻找一个实例所需要的另一个实例之服务，即获得某个实例的属性值或替第一实例完成一些处理。消息连接的表示如图 8.16 所示。

语音：

双箭头表明每个对象或分类结构都向另一个发送消息，“合法事件”需从“车辆”处获得实例值待计费时用；“车辆”需“合法事件”替它完成一些处理，比如“车辆”做修改时需重新计费。

消息连接标识完成之后,要在发送者的服务说明中建立消息连接的文档,在接收者的服务说明中建立相应的执行服务的文档。

详细说明服务。首先要为每个实例说明所需的外部可见行为,并给予特殊强调(语气时态上体现),然后使用规格说明模板建立需求框架,模板为系统设计提供设计文档。

至此,定义服务已基本完成。若想得到更完备的服务说明,可以通过增加支持表实现。支持表包括:

- ① 服务及可适用状态表,总结有关服务的适用状态信息;
- ② 关键路径分析,标志完成系统任务的关键的状态—事件—响应序列;
- ③ 时间和规模分析,对关键路径的第一个服务进行估计,由此得出整个路径的估计值,包括预计时间等数据。

本层的示意图见图 8.17。

系统分析阶段的工作全部结束。最终将所有文档汇集,包括 OOA 五个层次的系统模型示意图,属性及服务说明书,各种支持表(包括 DFD 图),可以得到从整体到部分的对系统数据,功能及动态特征的较深入认识。

## 二、面向对象的设计

OOA 是独立于程序设计语言的,OOD 作为 OOA 的进一步扩展进化,其初期在很大程度上依然独立于程序设计语言。此时最重要的工作是控制模型扩展过程产生的数据冗余,具体内容包括以下几个方面:

- (1) 属性值映射到表格,分类结构中的各组成部分都映射到一个表,每个对象亦做如此映射。
- (2) 选择关键码,作为系统控制下的唯一标识符。
- (3) 对数据规范化,采取折衷策略,允许一定冗余以避免表格膨胀。
- (4) 有一个独立的任务图,表示任务权限、多重例示以及通讯和协调。

在 OOD 模型中,对象或结构的右下角加标记指出任务名。若一个对象或结构的服务被划分在不同的任务中,则在每一服务后加一任务标识;如果一个服务被分割在多任务中,则把该服务名字扩充为不同名字,标识在任务图中相应位置。任务图与 OOD 模型之间是一一对应的。在设计阶段还要考虑一些细节,包括硬件体系结构(是否是分布式系统),软件体系结构(是否是多任务并行处理)等等。这些都将导致系统模型的扩张,其中最主要的是属性和服务的增加。最后在选取程序设计语言时应尽量选取支持面向对象各种机制的语言,使系统实现更方便快捷。经过分析与设计,得到有关应用领域解空间的完备信息,系统开发进入编码、调试阶段。试运行成功后整个系统就可投入使用。

## 本章小结

面向对象技术的出现给信息系统的发展带来了新的希望,面向对象技术最初起源于面向对象的程序设计语言,随着面向对象程序设计技术日趋完善,面向对象的思想及方法逐步成熟。系统开发人员通过面向对象的分析、设计及编程,将现实世界的空间模型平滑而自然地过渡到面向对象的系统模型,使系统开发过程与人们认识客观世界的过程保持最大限度的一致。利用面向对象开发方法得到的信息系统软件质量高,系统适应性强,系统可靠性高,系统可重用性和维护性好,在内外环境变化的过程中,系统易于保持较长的生命周期。总的来说,面向对象系统开发基本经历两个阶段:其一"WHAT",即研究问题域;其二"HOW",即如何实现目标系统。表现出来的特征及发展趋势就是:分析与设计更加紧密难分,程序设计比重愈来愈小(主要由于重用性提高),系统测试和维护简化,更易于扩充,开发模型愈加注重对象之间交互能力的描述。

Coad&Yourdon 的方法是一种循序渐进的方法，该方法通过标识类与对象、结构继承和组合、属性、服务以及主题构成面向对象的系统模型。

该方法实用且相对简单，模型易构造，但对系统动态特征表述不充分(主要是整体动态特征)，且反映系统整个功能特征的能力较差。但该方法反映系统结构完整，模型一致性好，易于完成开发系统，因而还不失是一种好用的方法。

应当指出，虽然面向对象开发方法是一种实用有效的系统开发方法。但不可否认的是，传统结构化方法依然有其可取之处，功能分解在把握系统整体一局部功能中是其他方法无可代替的，因此应有效吸取传统方法之长处，弥补面向对象开发方法之短处，使 00 开发方法更合理、更完善。