

第一节 软件工程与软件建模

詹文翰

zhanwenhan@163.com

课程安排

1

软件工程与软件建模

2

敏捷开发与项目协作

3

开发环境、规范与测试

4

需求、设计与程序框架

5

打包解包及其关键技术

6

压缩解压及其关键技术

7

加密解密与系统集成

8

软件发布与项目收尾

软件工程

软件工程方法学

软件危机

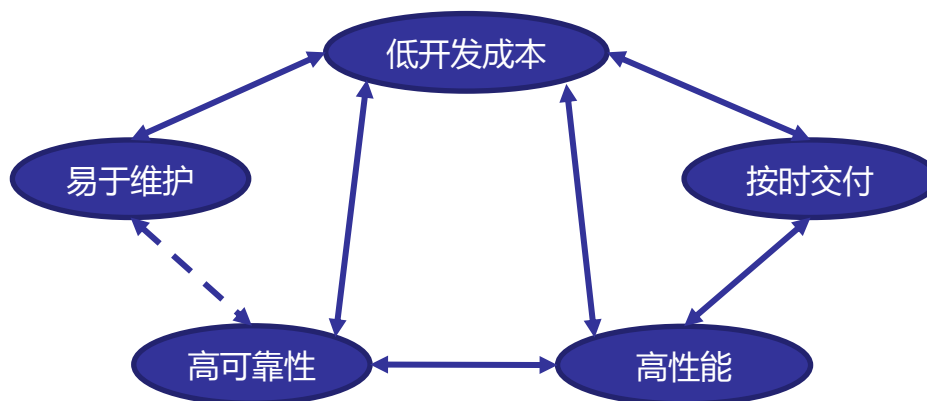
从20世纪60年代末至70年代初，“软件危机”一词在计算机行业广泛流传，指代在软件开发过程和维护中出现的大量问题。这些问题的产生都是由于落后的软件生产方式无法满足迅速增长的计算机软件应用需求所导致。

软件工程

“软件工程”的概念是1968年在德国召开的NATO（北大西洋公约组织）会议上首次提出的，强调应用工程化的原则和方法进行软件的开发和管理。

软件工程方法学

软件工程的目标



软件工程的三要素

软件开发过程：研究软件开发的生命周期，已提出了多种软件开发过程模型，如瀑布模型、原型模型、增量模型等。

软件开发方法：软件开发在分析、设计和编程的过程中所基于的方法学，主流的方法学包括：结构化方法和面向对象方法。

软件开发工具：为支持不同的软件开发过程和方法学而研制的计算机软件系统和开发环境。

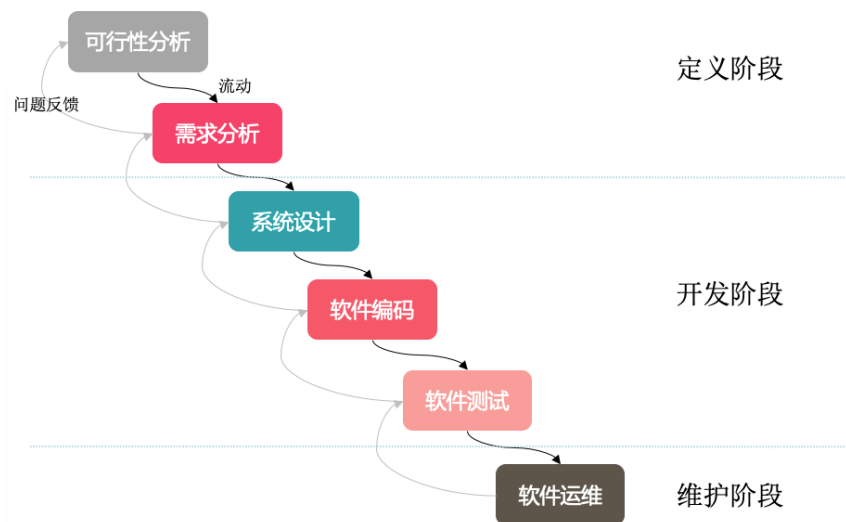
软件过程模型

瀑布模型

瀑布模型（Waterfall Model）是所有软件开发过程模型的基础。其是一个软件项目的理想开发架构，开发过程通过一系列的阶段顺序展开，从系统需求分析开始直到产品发布和维护。

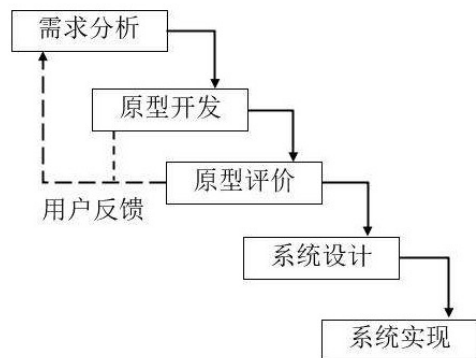
特点：

- 以里程碑的方式严格定义各开发阶段的**输入**和**输出**。如果上一阶段的输出达不到要求，则不开展下一阶段的工作。
- 各阶段定义清晰，强调文档，便于阶段评审、审计、管理和跟踪。
- 软件生命周期前期造成的Bug的影响比后期的大的多。
- 无法解决软件需求不明确、不准确和完善性问题。

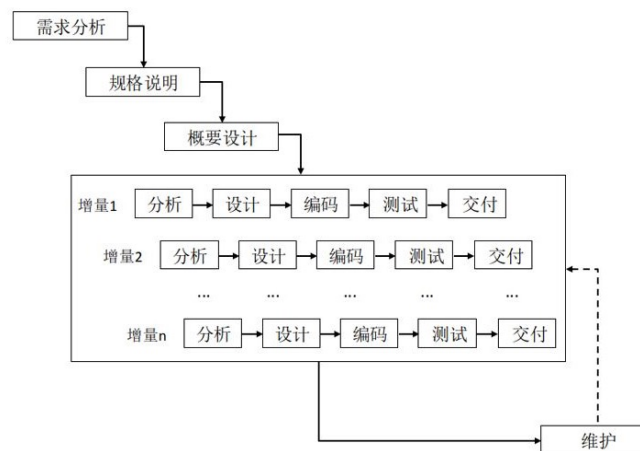


软件过程模型

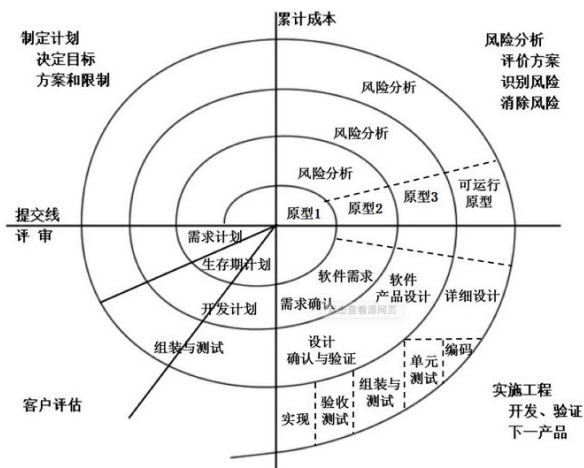
原型模型



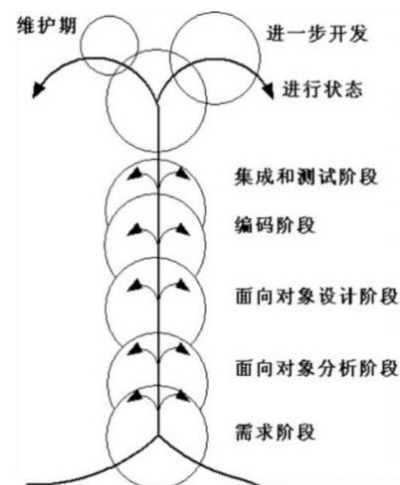
增量模型



螺旋模型



喷泉模型



软件开发方法

结构化方法

最受广大软件工程师所熟知的软件开发方法。该方法是20世纪60年代末在结构化编程的基础上发展而来的，由结构化分析、结构化和结构化编程三部分有机组成。

基本思想：自顶向下，逐步求精。通过把一个复杂问题（系统）进行抽象，将大问题（系统）分解为小问题（子系统），再将小问题（子系统）分解为更小的、相对独立的问题（任务）进行实现。

缺点：围绕“过程”来构造系统，而非“功能”本身；而用户的需求变化主要是针对功能的。因此，需求变化所导致的软件实现代价过大。

软件开发方法

面向对象方法

建立在“对象”概念基础之上，将面向对象的思想应用于软件开发过程中。该方法起源于面向对象的程序设计模式，包括**面向对象分析**、**面向对象设计**和**面向对象编程**三部分。

基本思想：**封装**、**继承**和**抽象**。客观世界由对象组成。将系统中的任何事物都抽象为对象，使开发软件的过程尽可能接近人类的自然思维方式。

由于客观世界的对象之间的关系相对比较**稳定**，因此，面向对象分析和设计的结果也比较稳定，从而在一定程度上提高了软件的**生产率**、**可靠性**、**易重用性**和**易维护性**。

面向对象方法正在逐步成为计算机领域软件开发的主流方法。

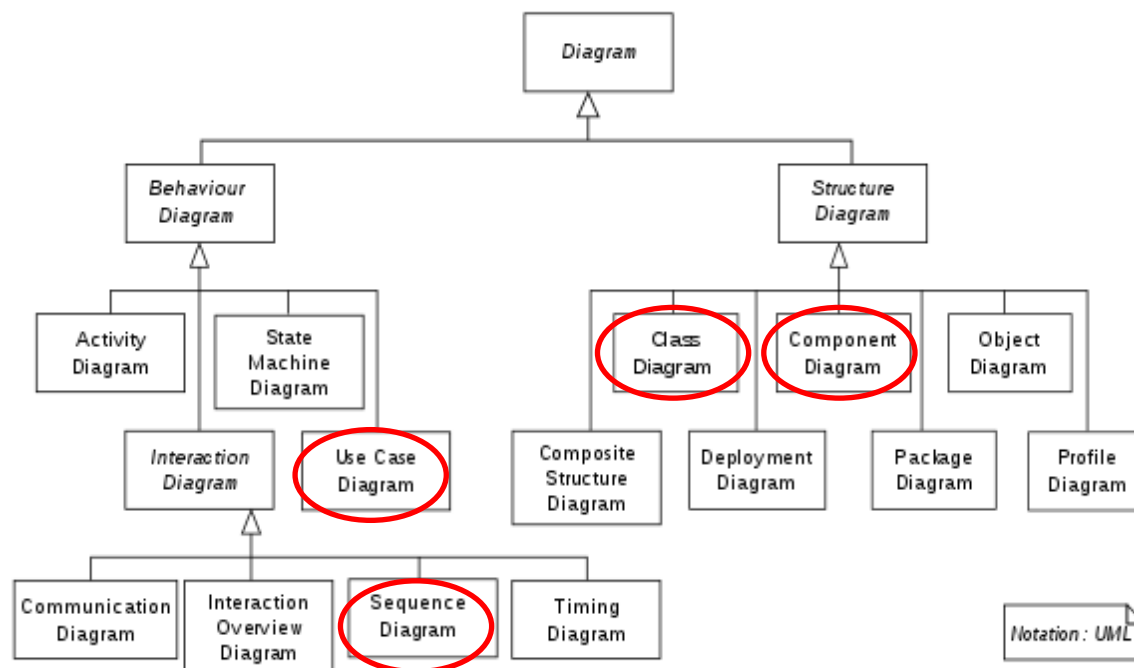
软件建模

统一建模语言UML

面向对象分析与设计的有力工具

UML产生于20世纪90年代中期，是一个通用的、可视化的标准建模语言，用于对使用面向对象方法所开发的系统进行说明。

UML是一种形式化语言，定义了一系列的图形符号来描述软件系统。这些图形符号有严格的语义和清晰的语法，可以同时为系统的静态结构和动态结构进行建模。



面向对象的需求分析

面向对象需求分析的核心是用例模型。用例模型是一个包含参与者、用例及它们之间关系的模型，描述了系统应为其用户做些什么以及在哪些约束限定之下的模型。

参与者：指系统以外的，使用系统或与系统产生交互的人或者实体。



用例：一个完整的、可以被参与者感受到的系统功能单元。



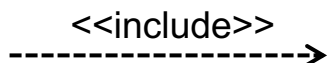
面向对象的需求分析

关系：关联、包含、泛化、扩展四大关系。

关联关系：参与者与用例之间的通信路径。



包含关系：在基础用例之上插入附加行为；支持在用例模型中复用功能。



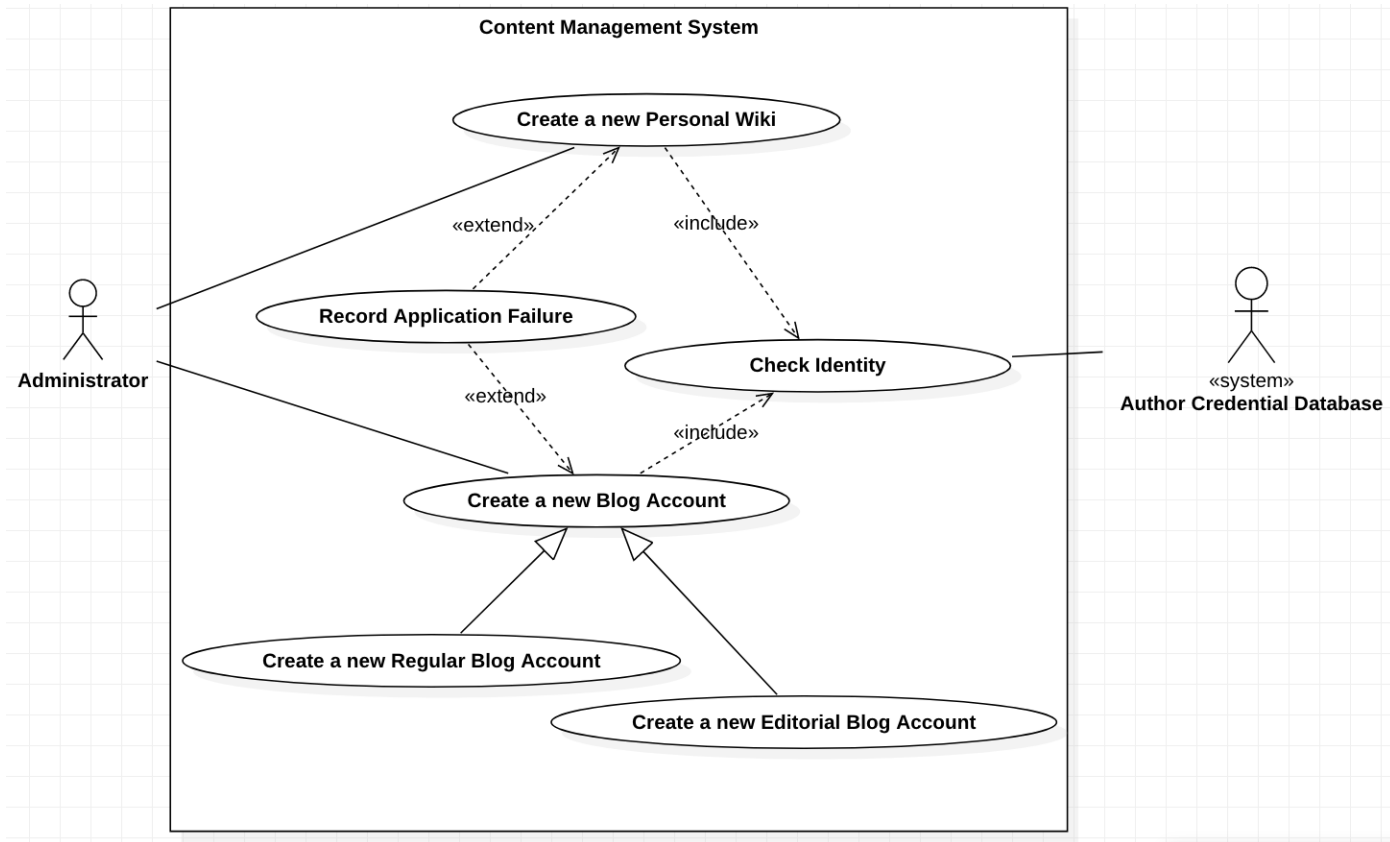
泛化关系：指示子代将接收父代中定义的所有属性、操作和关系，并可以增加新的特性。



扩展关系：对基础用例的功能进行可选的扩充。



用例图举例



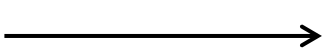
内容发布管理系统（出自《UML学习指南》）

对于关系的进一步说明

包含关系与扩展关系的区别：  

1、**include**通常发生在“多个用例中有可以提取出来的公共部分”，类似于提取公因式。包含方知道有被包含方（箭头指向方）的存在，但被包含方不知道包含方的存在。

2、**extend**表示扩展用例可以扩展被扩展用例（箭头指向方）在某方面的行为，被扩展用例需要留出扩展接口，交由扩展用例来实现。扩展方知道被扩展方的存在，但被扩展方不知道扩展方的存在。

依赖关系： 

正确的用例图是不存在依赖关系的。

依赖关系说明用例之间存在流程上的相互覆盖。针对这种情况，应将用例进一步细化，提取出公共部分，然后使用包含关系（**include**）来表示用例之间的关系。

用例描述

在进行需求分析时，仅用例图并不能提供足够的需求细节，必须假以用例描述，才能让客户以及设计者充分理解真实的需求。

用例描述举例（接上例）：

用例标识	xxx	用例名称	Create a new Regular Blog Account
创建人	xxx	创建日期	2020年11月2日
语境目标	一位新的或现有作者通过线下向管理员请求一个新的普通博客账户		
前置条件	作者需有适当的身份证明		
成功的结束状态	成功创建该作者的新普通博客账户		
失败的结束状态	该作者的新普通博客账户申请被拒绝		
参与者	Administrator		
触发器	管理员要求CMS建立一个新的普通博客账户		
基础用例	Create a new Blog Account		
包含用例	Check Identity		
扩展用例	Record Application Failure		
事件流	基本流程	1、管理员要求系统建立一个新的博客账户	
		2、管理员选择普通博客账户类型	
		3、管理员输入作者的详细数据	
		4、检查该作者的详细数据	
		include::Check Identity	
	扩展流程	5、成功创建新的博客账户	
		4.1、不允许该作者建立新的博客	
		4.2、博客账户申请被拒绝	
		4.3、此申请失败情况被记录在该作者的历史数据里	
extend::Record Application Failure			
非功能需求	安全性		
补充规格说明书	无		

面向对象的软件设计

需求分析过程 —————→ “做什么”

软件设计过程 —————→ “怎么做”

软件设计是一个迭代的过程，通过设计，需求被变换为构建软件的“蓝图”，即将软件分析模型中使用功能和行为来展示的需求信息转化为体系结构设计、数据或者类设计、接口设计和构件级设计等。

所有系统均可表示为两个方面：静态结构和动态行为。因此，面向对象的软件设计就是需要建立系统的静态模型和动态模型。

静态建模

静态模型描述系统的静态结构，是系统建模的基础，能够说明系统包含什么以及它们之间的关系，但并不解释系统中各个对象是如何协作来实现系统功能的。

UML中典型的静态建模工具：类图、对象图。

类图

类图是面向对象建模的主要组成部分，表示模型中的**类**、**类的内部结构**以及**类之间的关系**。其既可以用于程序的系统概念建模，也可以用于详细建模。

可以通过类图直接将模型转换成编程代码（**正向工程**）；也可以基于现有代码生成对应类图（**逆向工程**），用于辅助代码分析。

类的表示

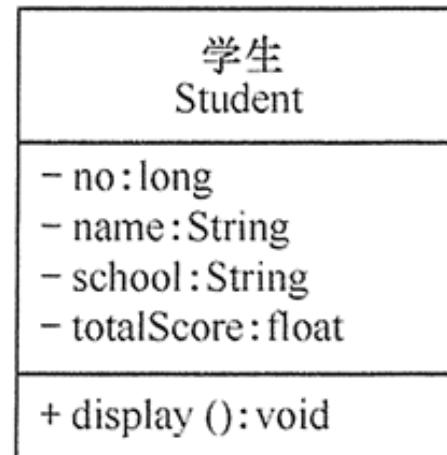
类的表示:

- (1) 类名 (Name) 是一个字符串。
- (2) 属性 (Attribute) 是指类的特性, 即类的成员变量。

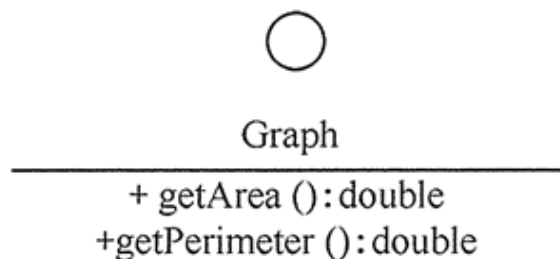
[可见性]属性名:类型[=默认值]

- (3) 操作 (Operations) 是类的任意一个实例对象都可以使用的行为, 是类的成员方法。

[可见性]名称(参数列表)[:返回类型]



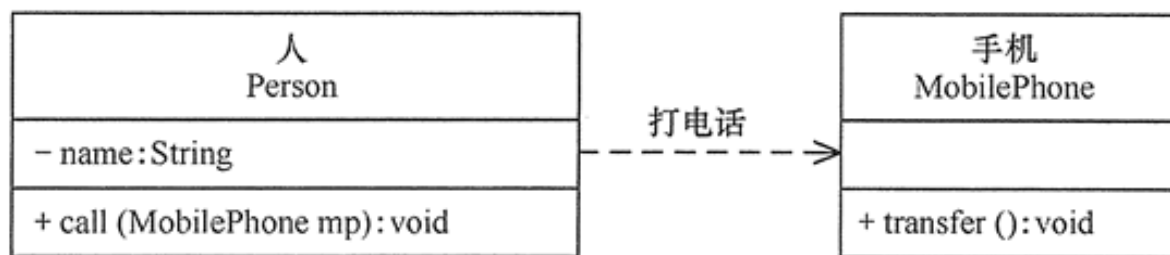
接口 (抽象类) 的表示:



类间关系

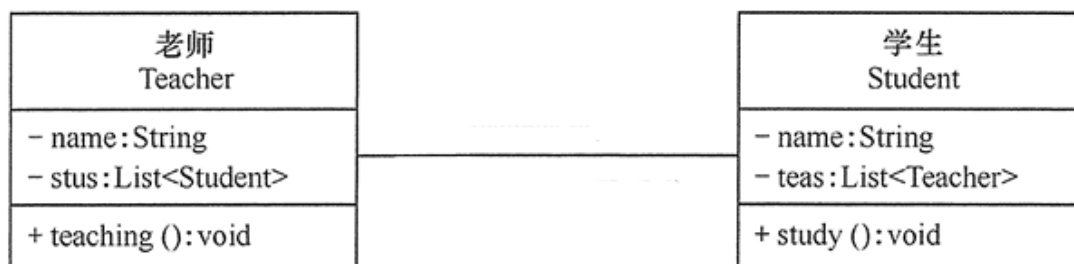
根据类与类之间的耦合度从弱到强排列，UML 中的类图有以下几种关系：依赖关系、关联关系、聚合关系、组合关系、泛化（实现）关系。

依赖（Dependency）关系是一种临时性的关联，大多表示“使用”。在代码中，某个类的方法通过局部变量、方法的参数或者对静态方法的调用来访问另一个类（被依赖类）中的某些方法来完成一些职责。

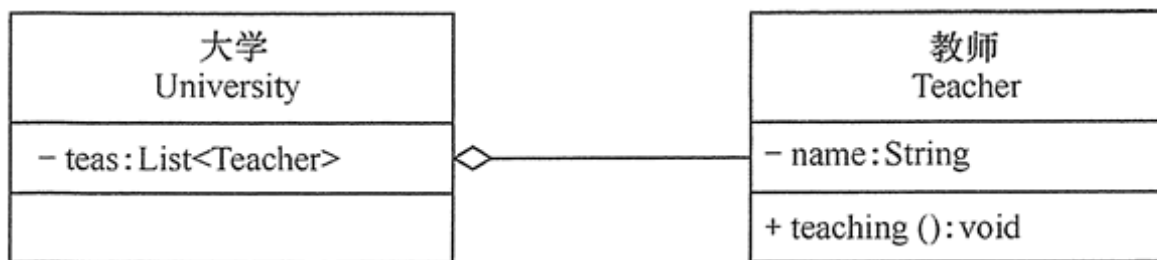


类间关系

关联（Association）关系是对象之间的一种引用关系，用于表示一类对象与另一类对象之间的联系，如老师和学生、师傅和徒弟、丈夫和妻子等。

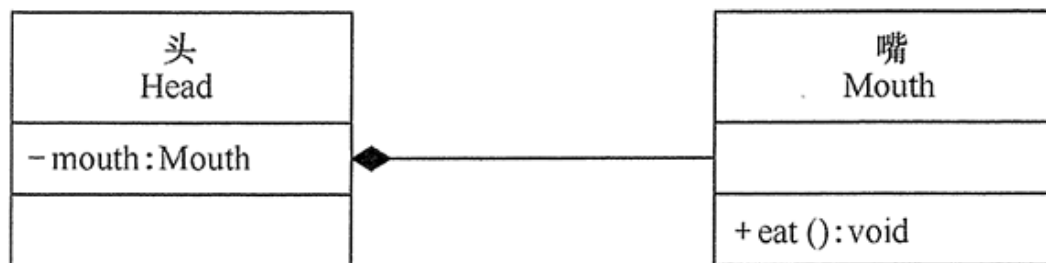


聚合（Aggregation）关系是关联关系的一种，是强关联关系，是整体和部分之间的关系，是 **has-a** 的关系。

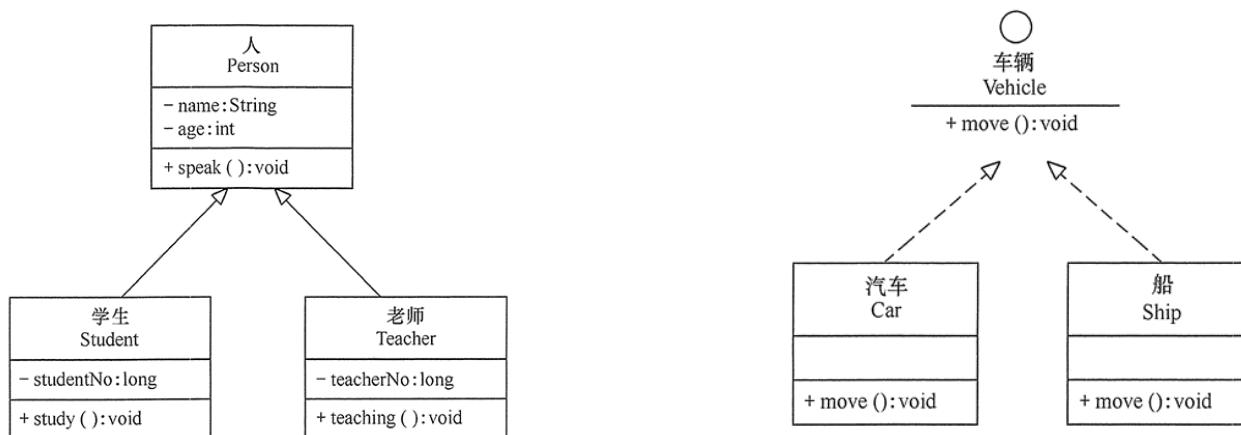


类间关系

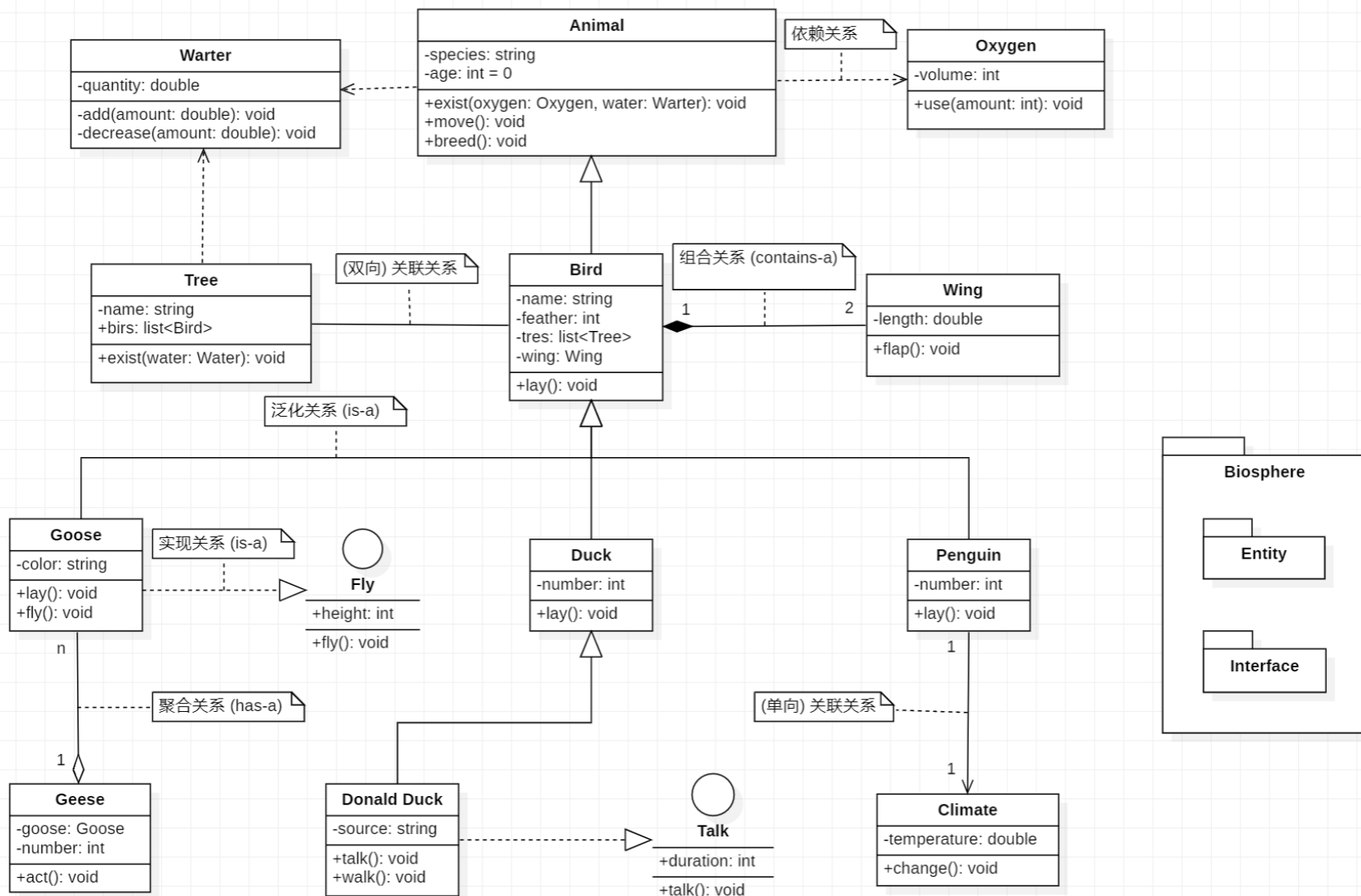
组合（**Composition**）关系也是关联关系的一种，也表示类之间的整体与部分的关系，但它是一种更强烈的聚合关系，是 **contains-a** 关系。



泛化（**Generalization**）关系或实现（**Realization**）关系是对象之间耦合度最大的一种关系，表示一般与特殊的关系，是父类与子类之间的关系，是一种继承关系，是 **is-a** 的关系。



类图举例



动态建模

任何系统功能均是通过系统结构元素之间的交互来完成的。动态建模描述了使用系统功能的具体过程以及各对象间的交互关系。结构元素（对象）之间通过通信协作或者自身状态改变的方式来实现系统的功能。

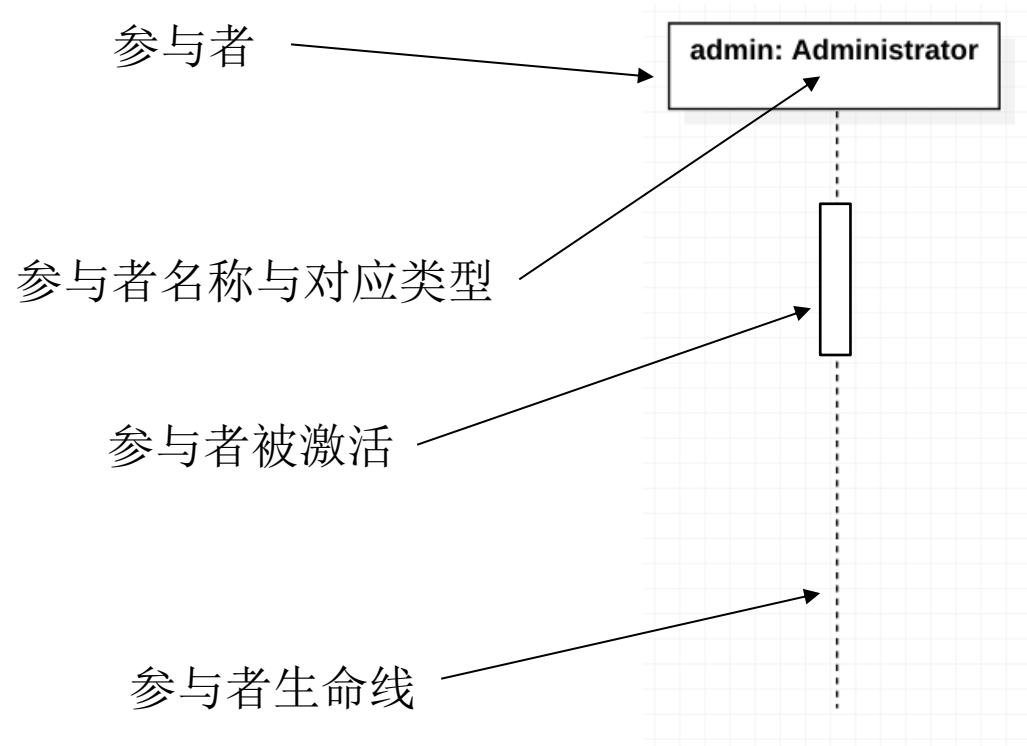
UML中典型的动态建模工具：交互图（顺序图、时序图、通信图）、状态机图、活动图（需求分析）。

顺序图（序列图）

顺序图用于描述执行特定用例时会触发哪些系统结构元素之间的交互，以及这些交互是以何种次序发生的。

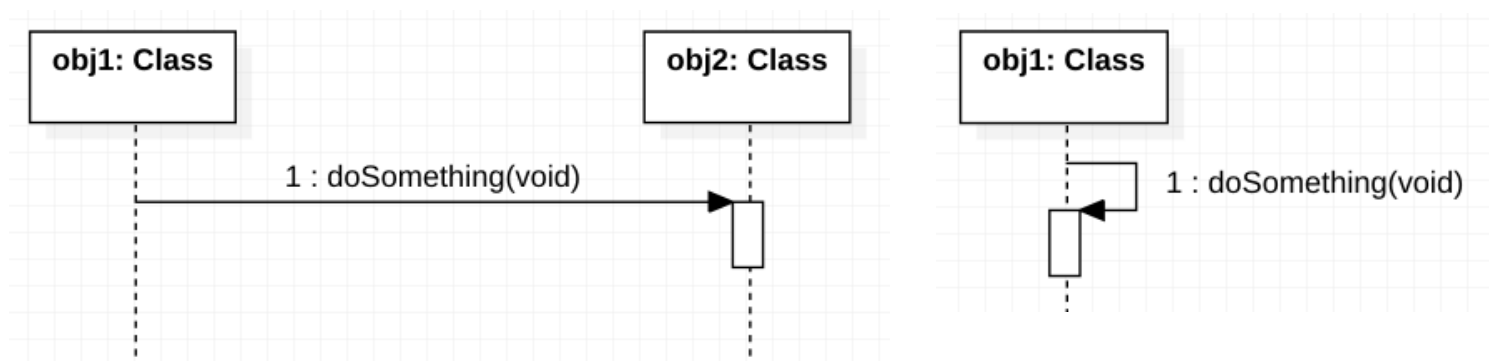
顺序图由参与者和消息构成。

参与者表示

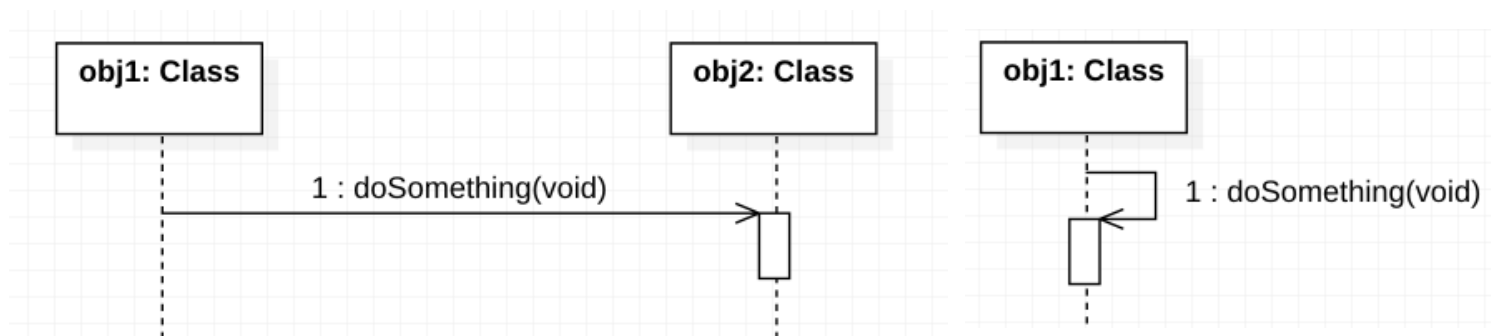


消息表示

同步消息：

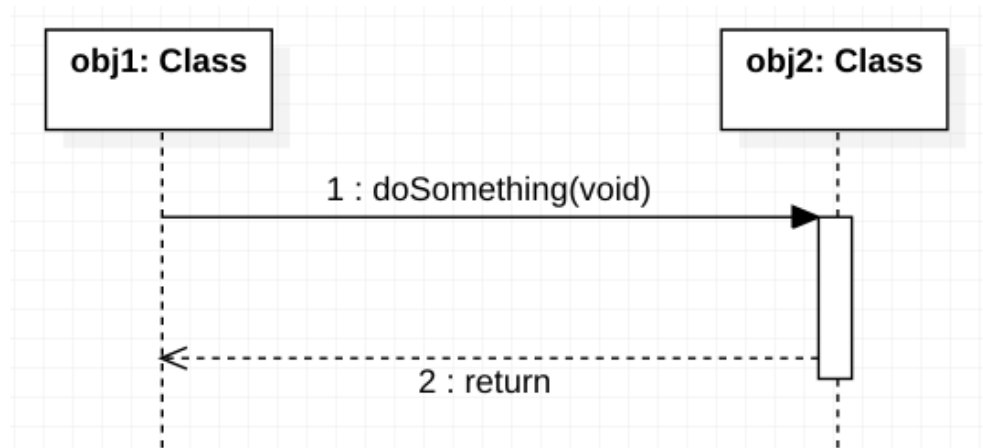


异步消息：



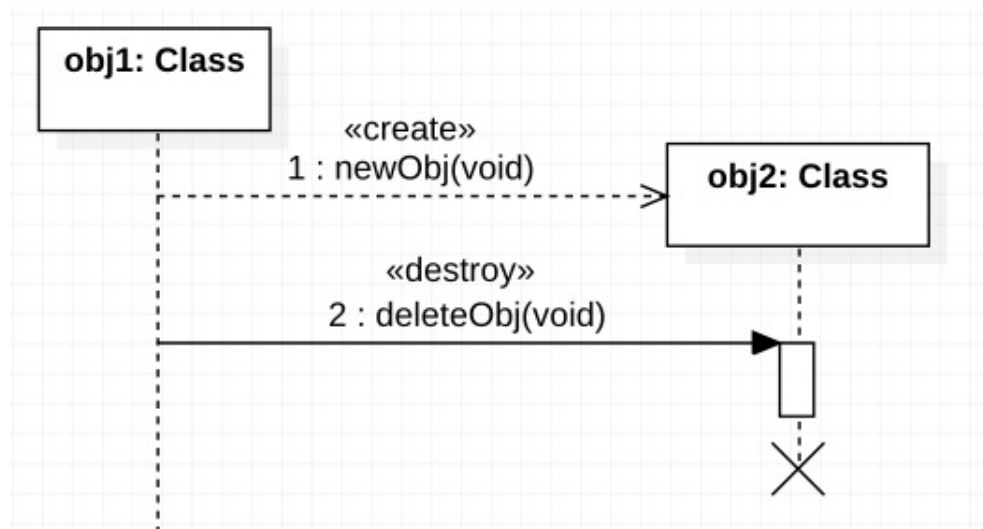
消息表示

返回消息:



参与者创建消息:

参与者销毁消息:

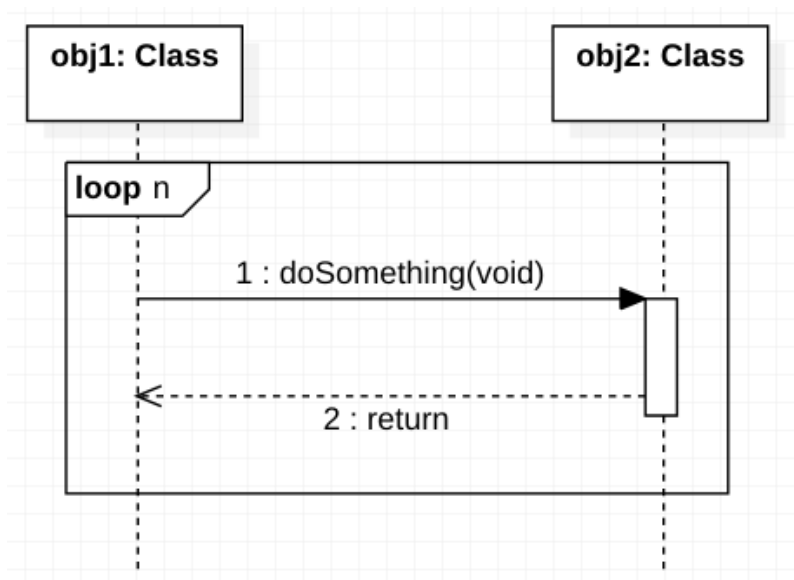


顺序片段

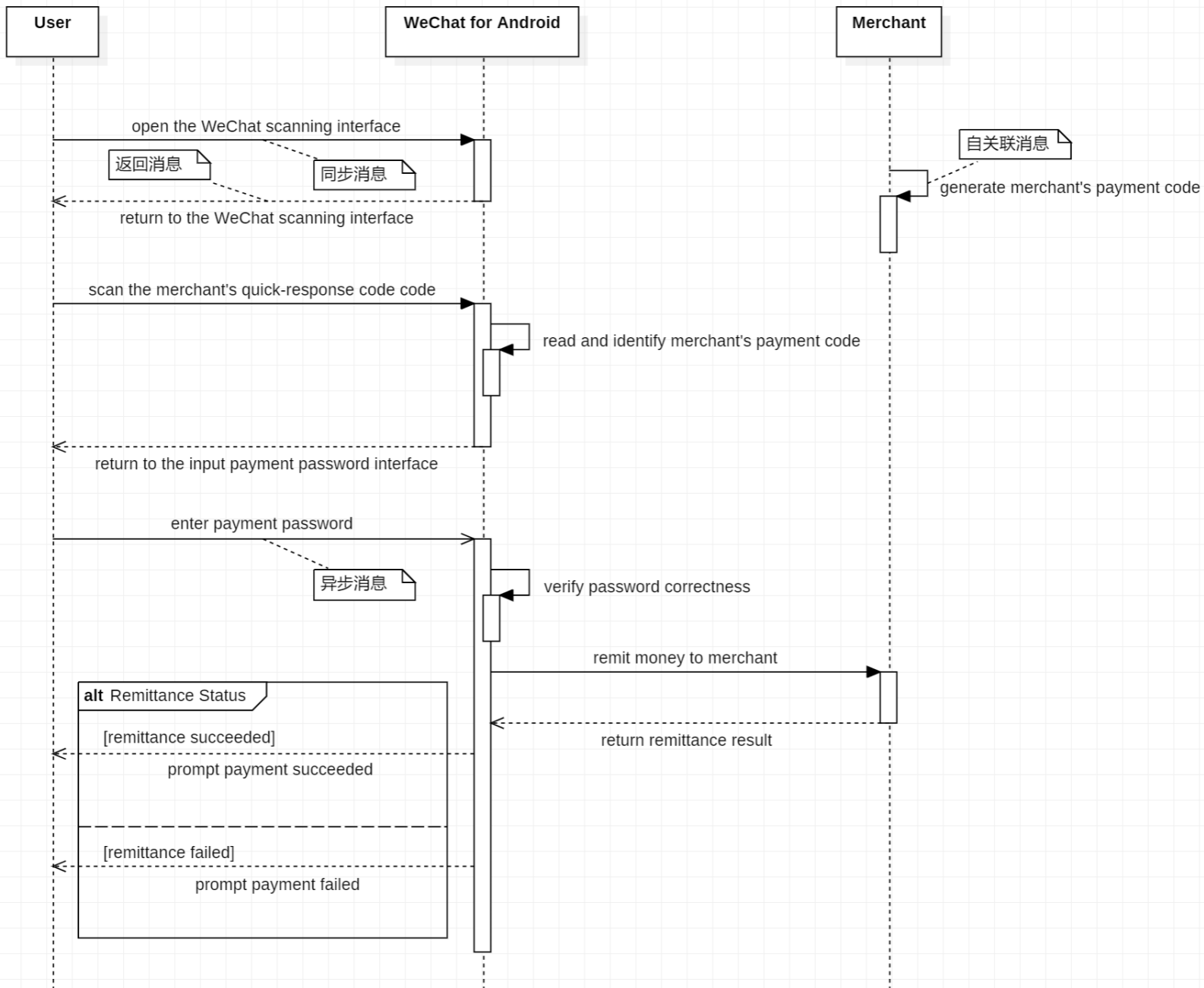
顺序片段可以让顺序图表达出复杂的过程逻辑。

常用的顺序片段包括：

片段类型	参数	作用
ref	无	引用其他片段
loop	循环条件	循环执行片段
alt	多选一	if-else选择
opt	可选	if选择



sd SequenceDiagram



物理体系结构建模

静态建模和动态建模是按照**逻辑观点**对应用领域的**概念建模**，而物理体系结构建模则是对应用自身的**实现结构建模**，提供了将系统中的类**映射成**物理构件和节点的机制。

UML中典型的物理体系建模工具：**构件图**（组件图）、部署图（分布式系统）。

构件图

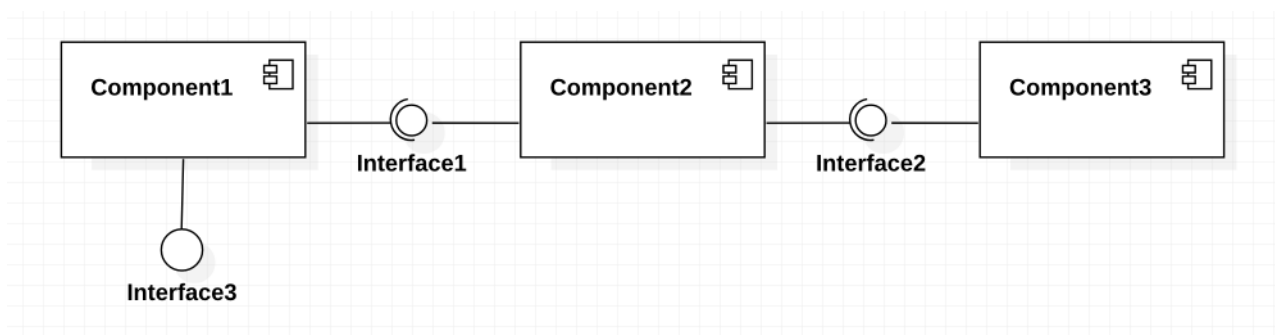
UML规范将构件（组件）定义为“系统中的一个定型化的、可配置的和可替代的部件，该部件封装并暴露一系列接口”。最常见的构建表现形式：动态库和静态库。

构件图（组件图）描述系统中的**构件**、**接口**及构件之间的**依赖**关系，帮助用户理清构件之间的相互关系，便于利用这些依赖关系来评估构件的修改对系统所造成的影响。

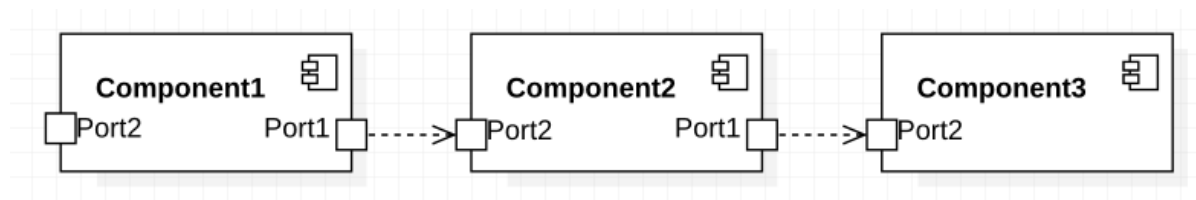
构件图表示法

构件图的表示方法还不是很统一。

第一种表示法：



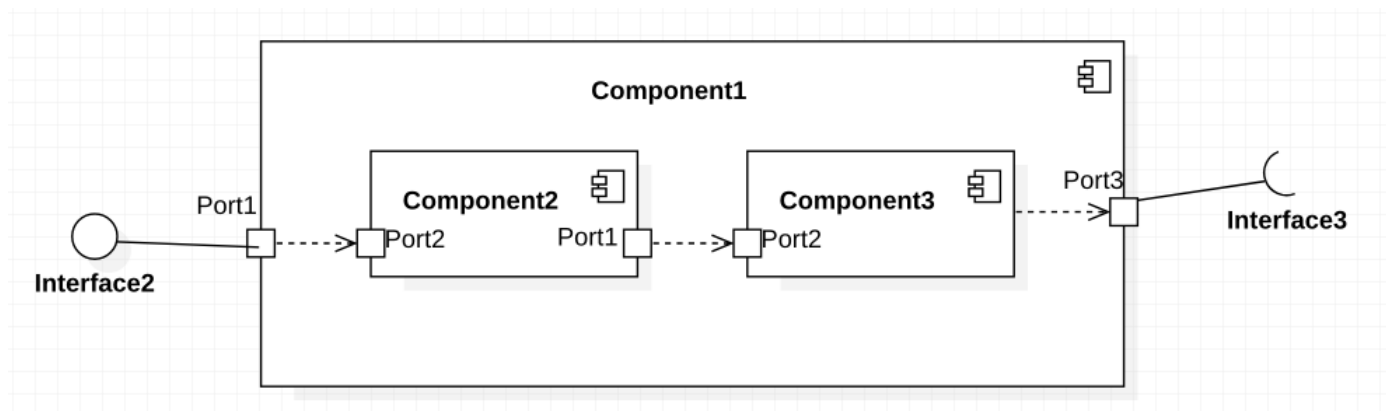
第二种表示法：



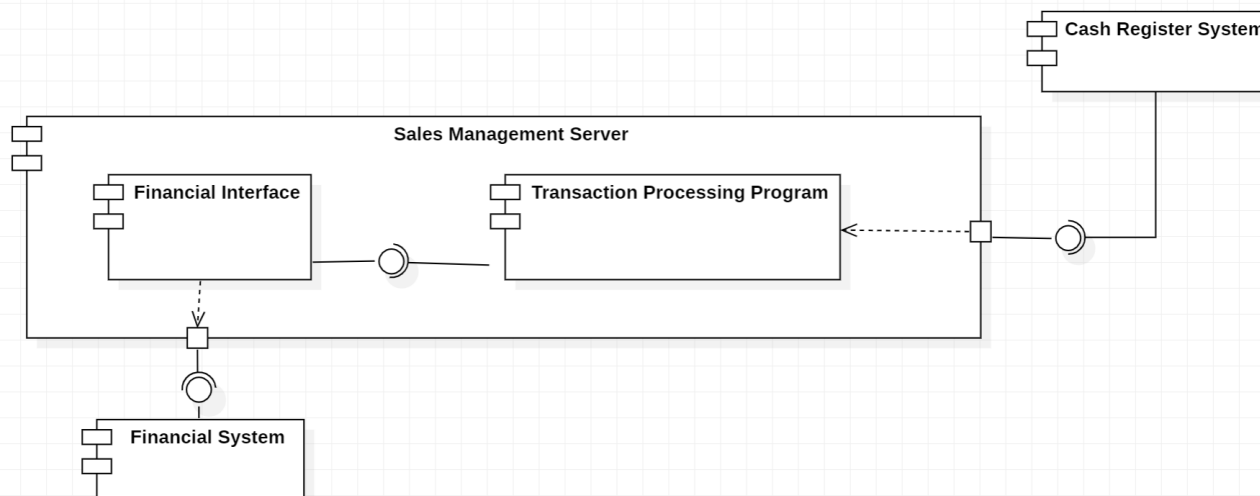
黑箱构件图与白箱构件图

上面的例子均表现为黑箱视图，其仅显示构件的外观，不指定构件内部实现。黑箱视图擅长显示系统的整体视图和构件依赖关系，通常包含多个组件。

白箱视图通常用于显示构件的内部实现，说明其由哪些类、接口和其它包含组件组成。白箱视图通常关注一个组件的内容。



构件图举例



Tips:

1、构划分原则应该是什么样的？

- 高内聚
- 松耦合

2、构件图应该在何时使用？

- 在系统概要设计时作为一个系统功能的初步划分
- 在系统详细设计时作为一个逻辑相对独立的程序或系统模块

实验环节

完成本课程实验二：

- 实验内容见《实验指导书2》
- 要求掌握starUML软件用法
- 不需要提交实验报告

对于选择Linux C++平台进行项目开发的同学，预习本课程实验一：

- 实验内容见《实验指导书1》
- 要求各小组构建好自己的开发环境，组内开发环境统一
- 不需要提交实验报告

项目要求：

- 至少使用用例图和用例描述完成项目的需求分析
- 至少使用类图完成项目设计的静态建模部分
- 至少使用顺序图完成项目设计的动态建模部分
- 至少使用构件图完成项目设计的物理体系结构建模部分