第三篇 工程实践篇

现代软件的开发日益流程化、正规化和系统化，这也被称之为“软件工程（化）”，软件工程相当于开发中的脚手架，它不仅能够帮助工程师们快速启动项目，减少枯燥的重复性工作，提升开发效率，而且提供了一套标准化、不易出错的开发过程。

本篇包括如何以图表强化软件协同，如何开发和调试出性能卓越的接口，如何以流水线的方式部署应用服务以及如何合理地运用开发过程提升软件质量。本篇内容是一次对软件工程实践的系统回顾，详细阐述了笔者多年来的软件工程实践经验和心得体会。

希望本篇内容能够将广大读者从“西绪福斯的苦役”中解脱出来。

第 11 章 字不如表不如图

俗话说“字不如表，表不如图”，相比于言语一大堆，用一张图显然可以让对方更快地明白自己所要表达的意思。开发中常用的图表无非是 UML 图、ER 图、架构图以及项目相关图表，本章就来一一盘点它们。

11.1 UML 图

UML 是“Unified Modeling Language”的英文缩写，称为“统一建模语言”，它提供了一套图形化的符号和规则，用于描述软件系统的结构、行为、交互信息，如同工程建筑行业中起着至关重要作用的蓝图一样，是一种设计“语言”。

UML 分为三大类：

1. 结构图：用于描述系统的静态结构，包括类图（Class Diagram）、组件图（Component Diagram）、部署图（Deployment Diagram）；

2. 行为图：用于描述系统的动态行为，包括用例图（Use Case Diagram）、活动图（Activity Diagram，也叫流程图）、状态图（State Diagram）、时序图（Sequence Diagram）、通信图（Communication Diagram）；

3. 交互图：描述系统中对象之间的交互关系，包括时序图（Sequence Diagram）和协作图（Collaboration Diagram）。

但实际开发中用得最多的是用例图、类图、流程图、状态图和时序图这五种。

11.1.1 用例图

所谓“用例（Use Case）”，可以把它简单地理解为一个独立且完整的用户功能需求，它用来描述用户与系统之间的交互和关联关系。通过用例，工程师可以知道：

1. 系统为谁服务？谁是最终用户（而非中间用户）？

2. 最终用户希望系统提供什么样的服务？又希望得到什么样的输出？

3. 最终用户能为系统提供什么样的输入？有哪些约束条件？

额外说明一点：在用例中务必明确谁才是最终用户。例如开发一个项目管理系统 PMS 时，可能会由一些业务方的主管、经理们提出具体的需求。他们虽然也是系统的用户，但不一定是最终用户，最终用户一定是那些每天都要接触并使用 PMS 的人：他们需要每天在 PMS 中提交日志，记录问题，汇报进度等，他们对于系统的需求和体验直接决定开发的成败。

用例图主要有四个组成部分：

1. 参与者（Actor）：参与者并非具体的人，而是指存在于外部并直接与系统交互的人、其他系统、子系统或对象。它在 UML 中用一个“小人”表示，如图 11-1 所示。

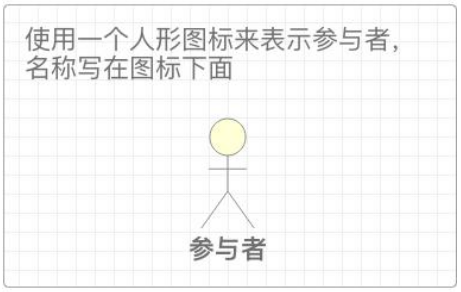

图 11-1 参与者 Actor

2. 用例（Use Case）：用来描述需要给参与者提供的功能或服务。它必须由参与者来执行，获取参与者的输入，并将输出结果反馈给参与者或系统，如图 11-2 所示。

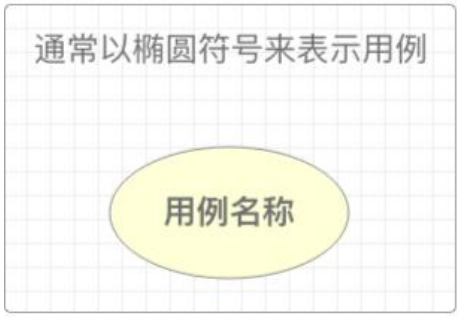

图 11-2 用例 Use Case

3. 关系（Association）：表示参与者和用例之间的联系，它也可以用来表示参与者和参与者、用例和用例之间的关系。常见的关系如表 11-1 所示。

表 11-1 用例图中常见的关系

关系类型	描述	表示符号
关联	参与者与用例之间的关系	——→
包含	用例之间的关系	<<include>> ⇢ 或 <<include>> ⇠
扩展	用例之间的关系	<<extend>> ⇢ 或 <<extend>> ⇠
泛化	参与者之间或用例之间的关系	——▷

4. 系统（System）：是用例需要描述的对象，它可以是一个软件平台或者一个硬件设备，

或者是一次促销活动，也可以是一个更大的系统的一部分。它有自己的边界，由参与者、用例和关系共同组成。

想画出完整的用例图，掌握以上内容就已经足够了。

另外，用例文档切忌啰嗦，一般都是非常简洁的“名词 + 动词”形式来描述业务主流程。例如，一个良好的用例文档就像这样表 11-2 这样：

表 11-2 良好的用例文档

步骤	描述
1	用户打开登录界面
2	用户输入用户名和密码
3	系统确认用户名和密码
4	用户登录成功

而比较啰嗦的反面典型可能会像表 11-3 那样：

表 11-3 良好的用例文档

步骤	描述
1	户打开浏览器，然后在地址栏中输入浏览器地址.....，浏览器确认 Cookie.....
2	页面显示出登录界面，用户单击页面输入框，然后输入用户名和密码
3	输入用户名密码时弹出人机确认对话框，用户确认身份，系统读取用户输入的信息
4	后台先验证用户名是否存在，核对用户密码是否正确。如果用户存在且密码匹配则用户登录成，否则登录失败，给出页面提示
5	.....

在画用例图时，只考虑需要实现什么功能即可，至于如何实现，暂时不必关心。

11.1.2 类图

类图描述的是系统中类的结构与类之间的关联关系，这是一种静态的建模方法，是对现实世界的抽象。类图分为两大部分：一是类的基本属性，包括类名、类的成员变量和类的方法，如图 11-3 所示。

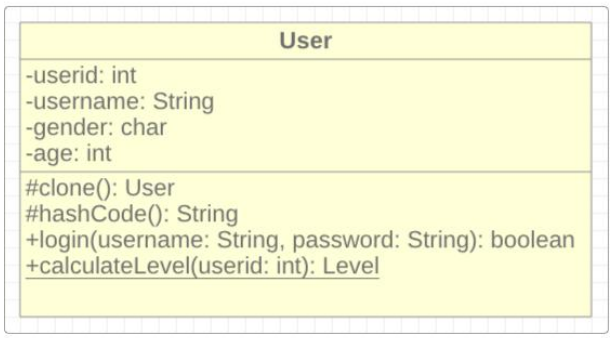

图 11-3 用户类的类图

1. +：表示 public，公共成员变量或方法；

2. -：表示 private，私有成员变量或方法；

3. #：表示 protected，受保护成员变量或方法；

4. 下划线：表示这是一个 static 静态成员变量或方法；

5. 斜体：表示抽象方法。也可以用两个尖括号表示抽象类，比如：<<抽象类>>。

类图的另外一部分就是类与类之间的关系，类之间有六种关系。

首先是关联关系，这是一种“拥有”的关系，它使一个类可以引用另一个类的属性和方法，如图 11-4 所示。

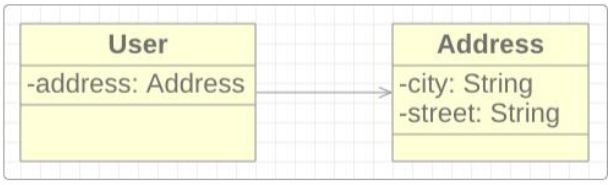

图 11-4 类的关联关系

上图展示的是一种单向依赖关系，如果类 A 关联类 B，且类 B 也关联了类 A，那么它们就是一种双向关联关系。

其次是聚合关系，这是一种特殊的关联关系，是一种比较强的关联。它描述的是整体和部分的关系，但部分离开整体依然能够独立存在，如图 11-5 和代码清单 11-1 所示。

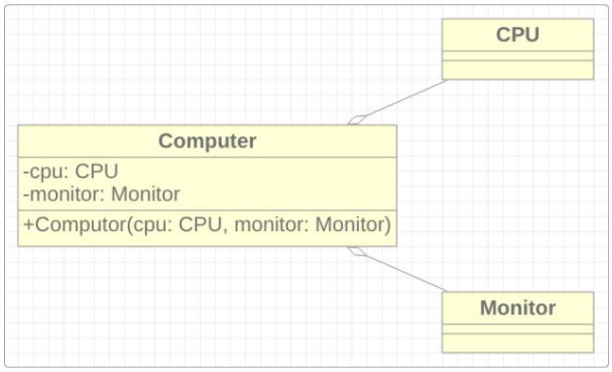

图 11-5 类的聚合关系

代码清单 11-1 类的聚合关系

```
public class Computer {
    private CPU cpu;
    private Monitor monitor;
    public Computer(CPU cpu, Monitor monitor) {
        this.cpu = cpu;
        this.monitor = monitor;
    }
}
public class CPU {
}
public class Monitor {
}
```

第三是组合关系，它和聚合关系类似，但比聚合关系更进一步，因为它规定了部分不能和整体分开。例如用户可以有积分和等级，但积分和等级离开了用户就没有单独存在的意义了，如图 11-6 所示。

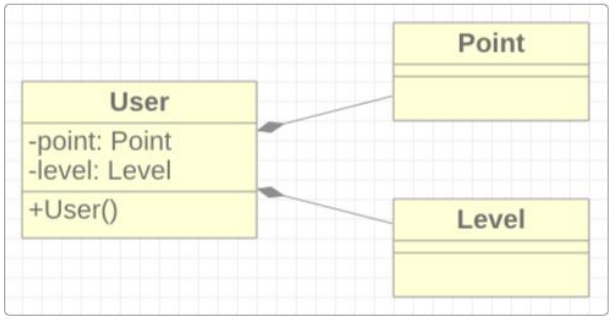

图 11-6 类的组合关系

再就是实现关系，它是指类实现某个接口。

既然有了接口的实现，那么肯定也会有类的继承关系，这就是 UML 类图中的第五种关系：泛化关系。

最后是一种是依赖关系，这种关系和关联有些类似，但比关联关系更宽泛，因为只要类 B 满足下面条件中的任意一个，就说类 A 依赖类 B：

1. 类 A 中有成员变量的类型是类 B；
2. 类 B 是类 A 方法的返回类型；
3. 类 B 是类 A 方法的参数类型；
4. 类 A 的方法中用到了类 B。

11.1.3 流程图

流程图是将各个业务环节和触发条件按照事件发生的顺序展现出来的一种图表。以用户登录为例，其流程图如图 11-7 所示。

图 11-7 带泳道的账号密码登录流程

在上图中，用黑色实心圆“●”来表示流程初始，而用“⊙”来表示流程结束。有三个竖长的矩形，分别是“用户”、“后端”和“前端”，这在流程图中称为“泳道”，就像泳池中的泳道一样，所以流程图有时候也叫“泳道图”。这些泳道代表一个个驱动业务开展的对象，它们都是事件或者业务规则的触发者。有的流程图并没有这些泳道，效果也是一样的。

11.1.4 状态图

状态图和流程图有些类似，但它关注的不是业务流程的变化，而是某些业务领域在事件发生时的状态变化，例如订单状态。有些 UML 图中将状态图弄的很复杂，包括诸如组合状态、历史状态、状态机等概念。一般稍微复杂一点的状态图会类似图 11-8 所示那样。

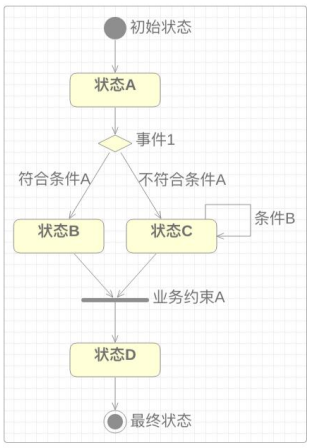

图 11-8 状态图的“模板”

和流程图类似，状态图中也用黑色实心圆“●”来表示初始状态，用“⊙”来结束状态。

其实大多数的状态图都没有这么复杂，因为搞得太复杂有可能不但达不到沟通的目的，反而会适得其反。以订单状态的变化为例，其状态图如图 11-9 所示。

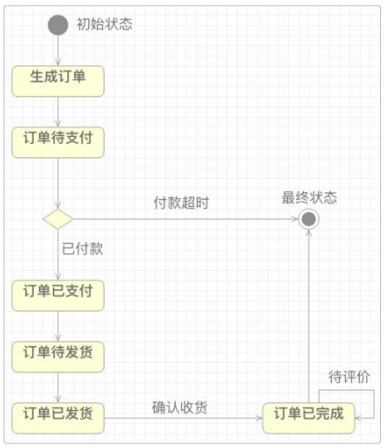

图 11-9 终端用户视角的订单履约状态图

11.1.5 时序图

时序图又被称为序列图（Sequence Diagram），它通过对象之间互相调用方法以及发送消息的时间顺序，来展示它们之间的协作过程。例如之前讲述权限相关知识时，就用一个时序图展现出了 OAuth 2.0 第三方授权登录的过程。

时序图既有对象的方法调用，也有调用时的流程展现。一个最简单的时序图包括四类组件：对象、生命线、关联消息和控制焦点，如图 11-10 所示。

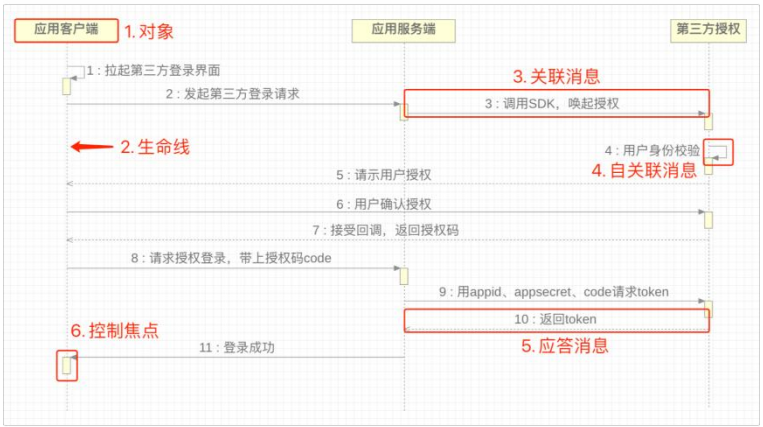

图 11-10 时序图的组成部分

生命线在时序图中表示为从对象图标向下延伸的一条虚线，表示对象存在的时间。当对象 A 调用对象 B 的方法时，就从对象 A 拉出一根实线箭头到对象 B，这就是关联消息，可以写上过程描述，也可以直接写上调用方法名，例如上图第 2 步的内容就可以换成“2: login()”。如果对象 A 是调用自身的方法，那么就有一个指向自身的箭头，表示自关联消息。而应答消息则类似于方法调用后的返回值。控制焦点又称为激活期，用来表示当前时间段内对象将要执行的操作，用一个竖长方条表示。

如果用时序图画出用户的网购过程，那么将会和图 11-11 相似。

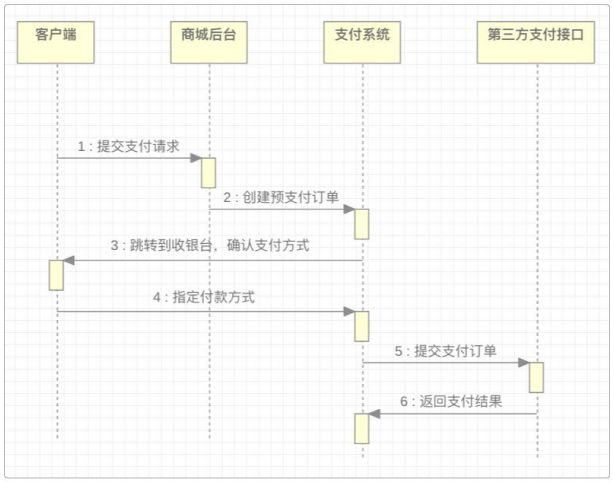

图 11-11 电商网站购物时序图

11.2 ER 图

ER 图是英文“Entity-Relationship Diagram”的直译，称为“实体-关系图”，它主要用于表示数据实体 Entity（通常对应于数据表）、属性 Attribute（通常对应于数据表中的字段）和实体之间的关系 Relationship（即数据表之间的关联），它是数据库设计中最常用的工具，没有之一。

11.2.1 逻辑模型

ER 图从概念上分为逻辑模型与物理模型，这两类模型本质上是同一个，只是详细程度不同。逻辑模型描述了数据之间的逻辑关系，它独立于具体的数据库管理系统 DBMS 或存储结构。逻辑模型通常更侧重于抽象的逻辑概念，而不涉及具体的物理存储细节。

ER 图由三要素组成：

1. 实体 Entity：可以是现实世界的对象，例如书籍、作者等；也可以是系统中的逻辑对象。比如消息、子系统、接口等。在 ER 图中，实体使用矩形框来表示；

2. 属性 Attribute：描述组成实体的要素，也就是数据表的字段；

3. 关系 Relationship：实体与实体之间的联系，而且还表示实体之间的数量关系。例如，一个出版社可以出版多本书籍，而一本书籍可以有多名读者，这就是简单的一对多关系。

常用的数据库客户端工具 Navicat 完全可以胜任 ER 图建模的工作，如图 11-12 所示。

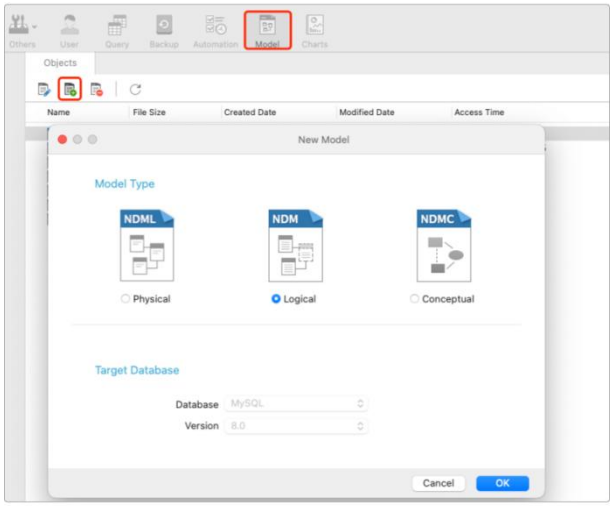

图 11-12 Navicat 的建模功能

Navicat 可以支持概念数据模型、逻辑数据模型和物理数据模型。所谓概念数据模型比逻辑数据模型更抽象，它只有实体及实体间的关系，而去除了属性，如图 11-13 所示。

图 11-13 Navicat 的概念数据模型

上图展示了两个简单的实体，即作者和书籍之间的对应关系：一个作者可以出版至少一本，也可以出版多本书籍。而一本书籍至少要有一个作者，也可以有多个作者。

在 Navicat 中有六种实体间的映射关系，这六种关系分别是：

1. None：表示没有任何对应的实体；

2. One and Only One：表示有且仅有 1 个对应的实体，如果双方都是这种关系，那就是一对一映射关系。例如，学生和学籍之间就是如此。Navicat 用图 11-14 来表示这种关系；

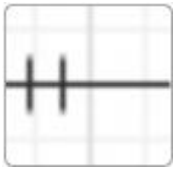

图 11-14 “One and Only One”映射关系

3. Many：有多个对应的实体。例如，1 个班级有多个学生。Navicat 用图 11-15 来表示这种关系；

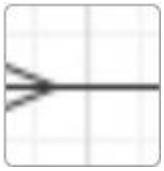

图 11-15 “Many”映射关系

4. One or Many：有至少 1 个，也可以有多个对应的实体。例如，1 本书籍至少要有 1 个作者，也可以有多个作者。Navicat 用图 11-16 来表示这种关系；

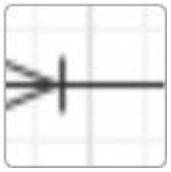

图 11-16 “One or Many”映射关系

5. Zero or One：有 0 个或 1 个对应的实体，也就是要么有 1 个，要么没有。例如，公民和身份证之间的关系。每个人在未成年之前都可以没有身份证，但成年之后最多只有 1 个身份证。Navicat 用图 11-17 来表示这种关系；

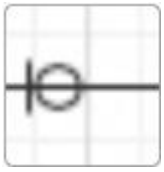

图 11-17 “Zero or One”映射关系

6. Zero or Many：有 0 个或多个对应的实体，例如用户和聊天群的关系就属于这种，某个用户可以没有加入任何聊天群，也可以在 1 个或多个聊天群中。Navicat 用图 11-18 来表示这种关系。

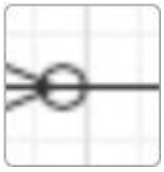

图 11-18 “Zero or Many”映射关系

至于实体间的关系到底应该是哪一种，这需要根据具体的业务规则而定。

11.2.2 物理模型

物理模型除了要添加更详细的字段属性值之外，它和逻辑模型没什么区别，而且从模型设计图的外观上也看不出什么不同，如图 11-19 所示。

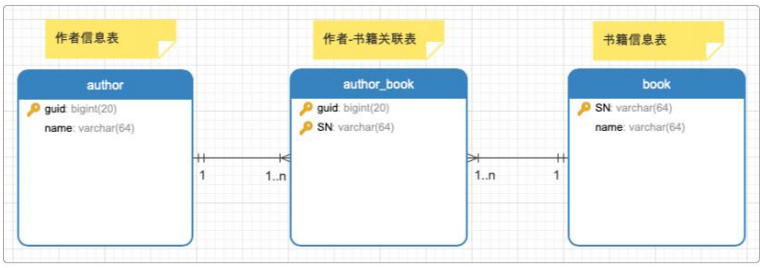

图 11-19 Navicat 中的物理数据模型

只不过在 PowerDesigner 中，物理模型是可以直接转换为 SQL 脚本的，所以不同的软件

工具在使用时，也会有不同的功能与便利。

11.3 架构图

相信读者们经常可以在一些需求文档、技术博客中看到比较漂亮的架构图形，例如图 11-20 所示的大数据系统架构图。

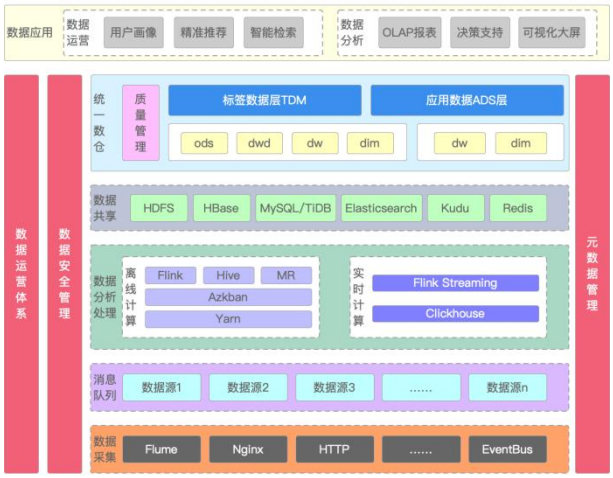

图 11-20 大数据系统架构图

下面来看看这些图是怎么画出来的。

11.3.1 架构思维

所谓的架构思维，其实是抽象思维、解构思维、集成思维、分层思维、发散思维、结构化思维、迭代思维、系统思维和模式匹配等诸多思维模式的一个有机融合。架构的核心作用在于将现实世界和抽象的 IT 实现之间连接起来，它最关键的一点是既要深刻理解业务，也要深刻理解技术，并由业务来驱动技术，使技术最终为业务服务，达到多方面的平衡，如图 11-21 所示。

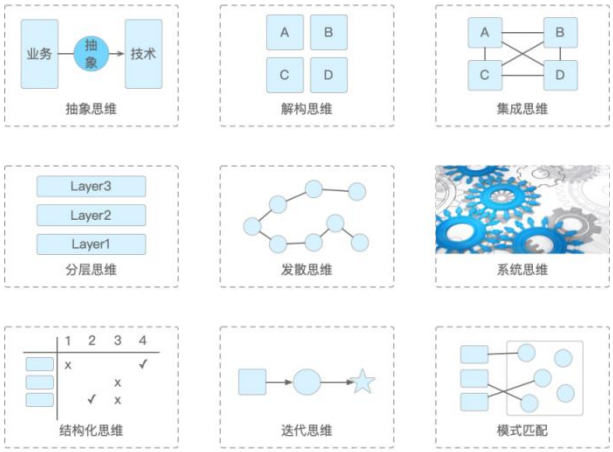

图 11-21 架构思维

虽然有这么多的思维模式，但好的架构设计都会经过“先抽象性解构，再系统性集成”的过程。抽象和解构是基础，能够对复杂的问题抽丝剥茧，再分而治之。同时，将分解后的各种部件按照“高内聚，低耦合”的原则，系统化地集成为一个有机的整体，这又是另一项了不起的能力。“抽象”、“解构”、“系统”与“集成”这几种思维模式一旦完美融合，就会产生如图 11-22 中所展示的那种“美”。

图 11-22 中国建筑中的榫卯结构

将复杂的业务逐步分解完成后，一个庞大的系统就已经被拆分成了很多个子系统，一个子系统本身又可以分为多个实现步骤或阶段，将这些零散的节点汇集和归纳起来之后，仿照组织结构的层级，将它们的功能职责划分为不同的层次后再组织起来，这就是分层思维。

分层思维在软件开发中得到了非常普遍的应用，从 C/S 架构，到 B/S 架构，再到目前的 MVC 和云服务架构（IaaS、PaaS 和 SaaS），分层可以说是无处不在。

架构在变，但架构的设计思维一直都没有变，那就是更加面向业务、更加适应复杂且频繁变化的业务。而且，不管什么思维，什么架构，什么模式，没有好坏之分，只有适用之别。

11.3.2 架构图模板

虽然学习架构思维需要时间的浸润和经验的积累，但直接套用模板也可以快速画出美观的架构图。就笔者的经验而言，常用的架构图包括但不限于如下几大类：

1. 规范辅助型模板：它将一个完整的架构图拆分为“左中右”式布局，两边小中间大。中间部分放的内容主体，如业务模块、技术栈等，而两边则是放一些具体的标准、规范和辅助工具，例如认证授权、技术标准、开发规范、数据安全等，当然也可以没有左边和右边的部分，如图 11-23 所示。

图 11-23 规范辅助型架构图模板

2. 上下分层型模板：它在规范辅助型模板之上，将中间的内容主体按照分层思维划分成了不同的层级，层级的划分可以参照云服务的标准架构，也可以按照具体业务需求和技术规约进行，如图 11-24 所示。

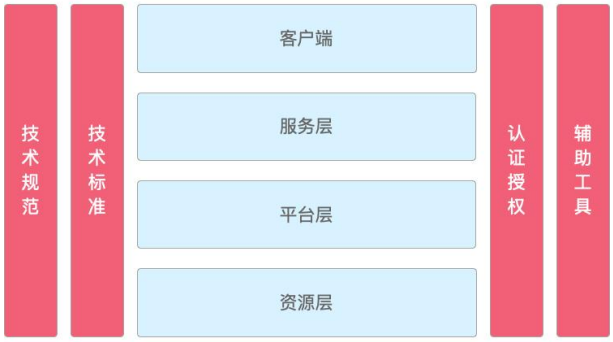

图 11-24 上下分层型架构图模板

3. 组件切分型模板：这种模板是纵向分层之后，再进行横向切分，将某一层或某几层切分为更细小的组件、服务或模块，这样看起来内容细节更加丰富充实，如图 11-25 所示。

图 11-25 组件切分型架构图模板

4. 融合复用型模板：它基于组件切分型模板，将其中某些重复部分或共性部分提取出来作为公共组件，并融合与之相关的其他业务板块，演变成为架构中台，如图 11-26 所示。

图 11-26 融合复用型架构图模板

5. 纵横交错型模板：这种架构图在一般的互联网和企业应用中比较少见，主要出现在系统集成应用中，它体现的是各个子系统中的关键部件，及子系统之间的连接关系。它把 UML 组件图和数据流图给结合到了一起，如图 11-27 所示。

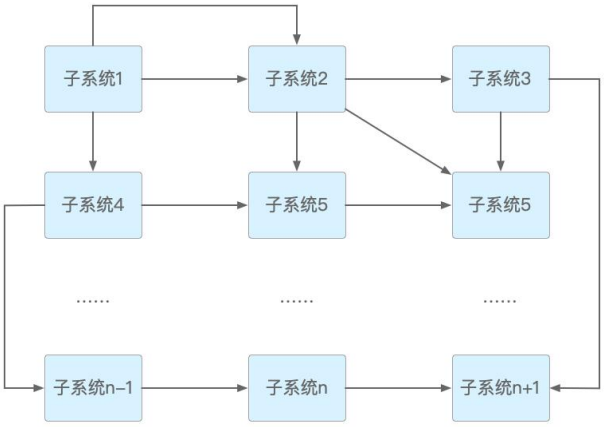

图 11-27 纵横交错型架构图模板

6. 网络拓扑型模板：在涉及到网络和硬件的应用中这种架构图属于标配，纯软件项目中基本看不到，但是出于完整性考虑，笔者还是将它在此归为一类，如图 11-28 所示。

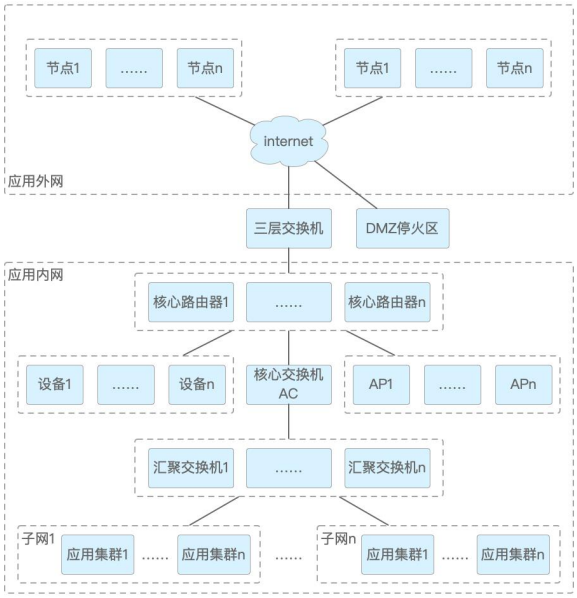

图 11-28 网络拓扑型架构图

不管是哪种模板，都是多种架构思维方法的排列组合，只有掌握了底层的架构思维方法，才能在画架构设计图时信手拈来。

11.3.3 架构图分类

俗话说“千人千面”，按照类别，架构图分为业务架构图、应用架构图、技术架构图和数据架构图这几个大类。

1. 业务架构图：是用来描述组织不同的业务活动、流程、信息和系统之间关系的图形。通过业务架构图，产品经理、项目经理或工程师们可以更清晰地了解组织的业务结构、运作方式和信息流动路径，有助于业务规划、流程优化和系统设计。图 11-29 展示的是一个典型的零售企业所拥有的业务形态和结构，也属于典型的组件切分型架构。

图 11-29 典型的零售企业业务架构图

2. 应用架构图：这是比较容易和业务架构图混淆的一类架构图，但在概念上它和业务架构图是不同的。如果说业务架构图关注业务流程和信息供给，那么应用架构图的关注点在于应用程序本身的设计原则、技术选型、通信方式及各组件之间的依赖关系。它的目标是确保系统的稳定性、安全性、可维护性、易用性和可扩展性，不过这两者之间常常会搞混。图 11-30 展示的就是一个微服务应用架构图，也属于典型的组件切分型架构。

图 11-30 典型的微服务应用架构图

3. 技术架构图：顾名思义，它是描述软件系统或应用程序的技术组件、模块、接口和

它们之间关系的图形。虽然技术架构图和应用架构图都有关于软件系统的依赖关系和通信方式，但应用架构图侧重于展示软件系统的功能模块、业务流程和业务逻辑，是一种高层次的设计，不关注技术细节；而技术架构图则侧重于描述软件系统的技术实现细节，包括各个模块的部署方式、通信协议、数据传输方式等。图 11-31 展示的是典型的微服务技术架构图。

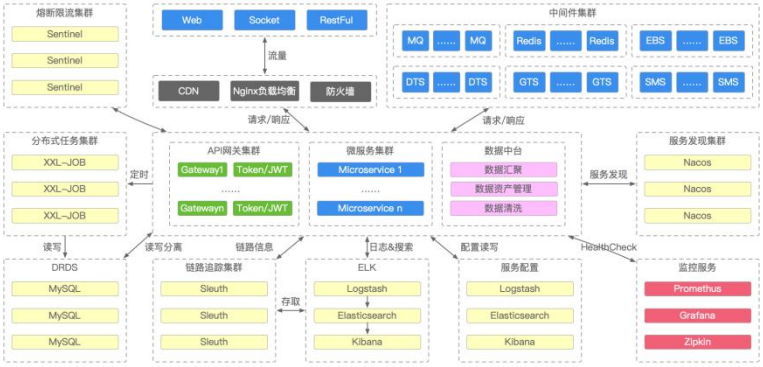

图 11-31 典型的微服务技术架构图

4. 数据架构图：所谓的数据架构图，就是要让业务数据、系统日志、人员信息、消息往来、资讯文档、会议纪要等各种各样的数据流向能够为企业创造价值的地方，同时也为企业的战略规则、设计和决策提供可靠的依据。数据架构并非指的是狭义上的数据系统或数据库设计，它和企业内全部的 IT 系统紧密相关，因为它才是企业信息化真正的灵魂，典型的数据架构如图 11-32 所示。

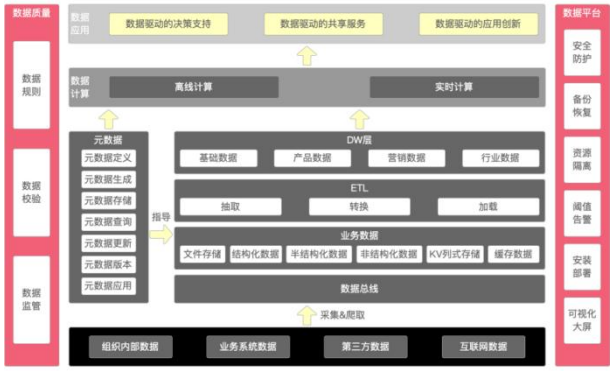

图 11-32 典型的数据术架构图

在很多技术文档或博客中出现的架构图，并不能够让人很明确地知道它们归属于以上的哪一个分类，这并不重要，重要的是它能否体现出架构设计者的思路，体现出一个可落地的技术实现，能够让人一看就明白，达到沟通、交流和持续改进的目的，才是关键所在。

11.4 其他

对于架构师来说，技术开发并非工作的全部内容，其间总会伴随着一些任务安排、制定计划、人员协调和进度报告的事项，这些安排、计划、协调和报告如果能以图表的形式呈现，不仅可以节约上级的时间，也能让其对全局一目了然，提升沟通的效率和效果。

11.4.1 甘特图

所有的项目在开始之前，都需要事先定义的清晰的目标和计划。没有这些明确设定的步骤、任务和截止日期，项目将有失控的风险，在项目管理中，最常见的一类图表就是甘特图（Gantt Chart），它是以提出者亨利·劳伦斯·甘特（Henry Laurence Gantt）的名字命名的。

甘特图不仅仅可以用于项目管理，实际上，只要是与时间、进度、成本相关的任务，都可以用甘特图来制定计划。对于个人工作安排来说，也可以通过甘特图来直观地设定，如图 11-33 所示。

图 11-33 个人工作计划甘特图

画甘特图的工具很多，Windows 中的 Project、Visio、Excel，MacOS 中的 OmniPlan，以及一些网络应用，如 ProcessOn、Ganttable 等，可以选择一种自己熟悉的工具软件使用。但工具是无法代替思想的，只有具备了良好的项目管理思维才能轻松画出好的甘特图。

以单身一族做饭为例，其周末烹饪午餐的过程可能会如图 11-34 所示的那样。

图 11-34 单身一族做饭甘特图

对于个人就餐而言，“蒸饭”和后续的“摘菜”、“洗菜”和“炒菜”是并行的，但如果将“做饭”这件事换成一个专业团队来干的话，其“项目”的执行过程就完全不一样了，以笔者对餐饮行业的肤浅了解，如图 11-35 所示。

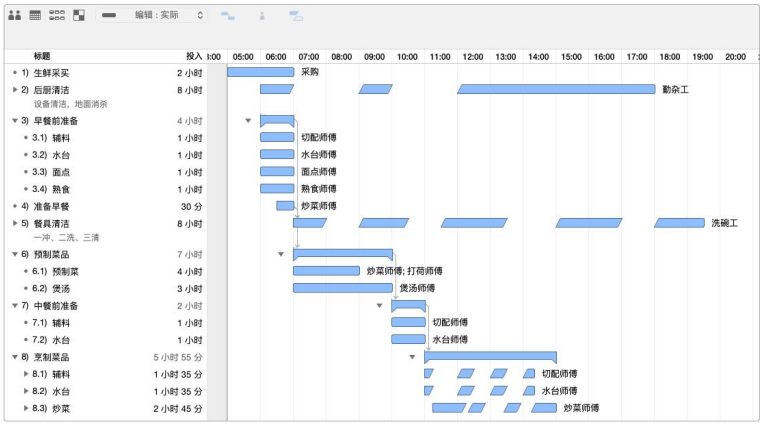

图 11-35 专业团队烹饪甘特图

从图中可以看到，无论是做饭内容还是流程都完全不一样了。例如对于勤杂工和洗碗工来说，其工作内容并非连续的，他们不会一直拖地，也不会一直洗碗，而是有需要的时候才拖地洗碗。在制作午餐的烹制菜品时，切配、水台和炒菜等各路师傅的工作也是穿插进行的，不像软件开发中某个功能的开发可以由某位工程师一直持续进行。

另外，上图中的关键资源就是切配、水台和炒菜这三个岗位，这类似于开发团队中的骨干工程师，他们的工作效率可以成为制约项目的瓶颈之一，而且也是项目的关键路径。

笔者强调过，甘特图只是项目管理思想的一种体现，只要具备了成熟的项目管理思想，即使没有画图工具，“甘特图”也会自然而然地出现，如图 11-36 和图 11-37 所示。

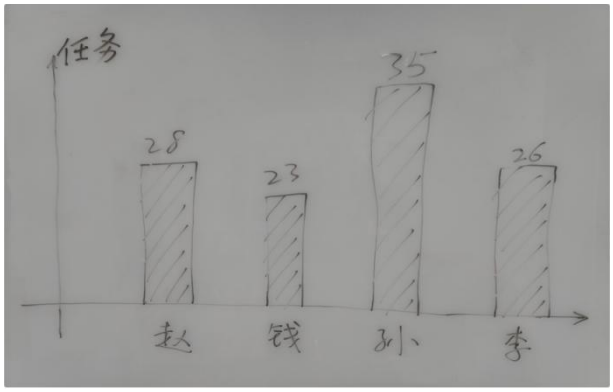

图 11-36 任务分配图

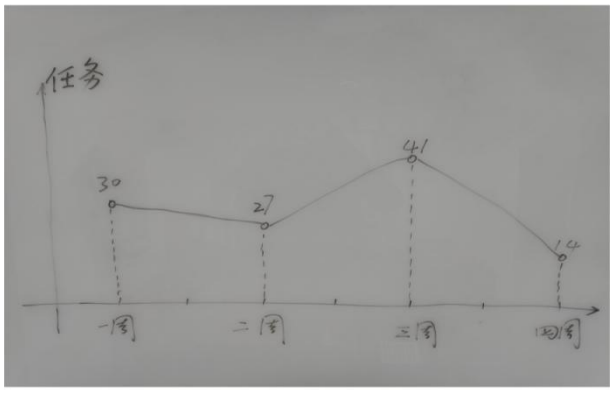

图 11-37 任务进度图

从上面两幅图可以看到，在四周的时间内，赵钱孙李四位开发工程师总共完成了 112 个开发任务，每个人完成的开发数量和每周总共完成的开发数量也都明确地展示了出来。其中，由于前两周属于技术攻坚阶段，因此每周完成的任务数较少。而第四周由于是收尾，所以完成的任务数不多，更多地是测试和修改 BUG 的工作。

所以，即使没有画图工具，拿一张 A4 纸同样可以清晰地记录下这些数据，真正重要的是项目管理的思维方式和意识，它们才是甘特图的灵魂。

11.4.2 鱼骨图

另一种常见的图形为鱼骨图（Fishbone analysis method），因为画出来的图形像鱼骨而得名。由于鱼骨图是由日本管理大师石川馨发展出来的，所以又叫石川图，鱼骨图的结构如图 11-38 所示。

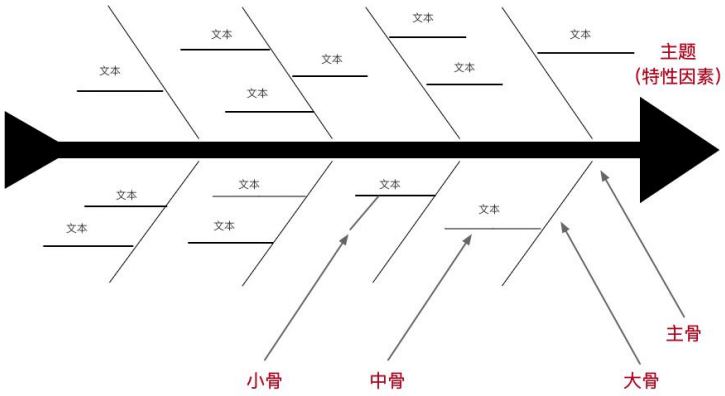

图 11-38 鱼骨图的结构

鱼骨图有三种类型，也是用来处理三种类型的问题：

1. 问题型：可以称之为“what”，用于定义问题；
2. 原因型：可以称之为“why”，用于找到产生问题的根本原因；
3. 对策型：可以称之为“how”，即找到如何解决问题的办法。

这几种不同类型的鱼骨图之间没有绝对的界限，图 11-39 就是以鱼骨图的形式做的一份半年工作总结。

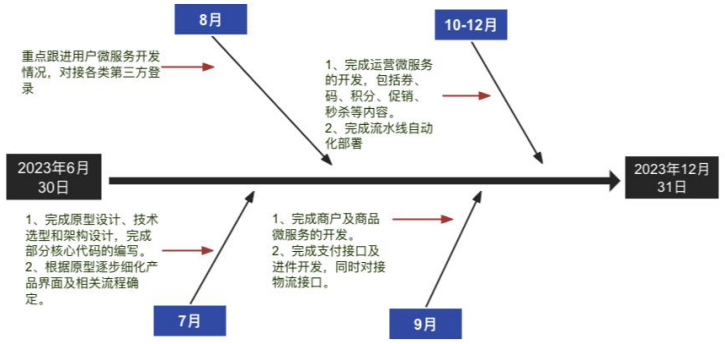

图 11-39 鱼骨图用于工作总结

当然，鱼骨图的强项还是在问题定义和寻找对策上，如图 11-40 所示。

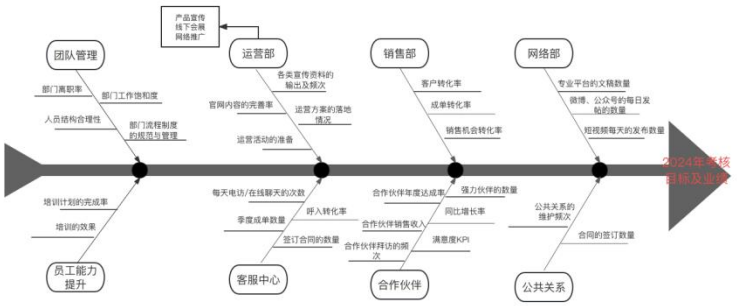

图 11-40 鱼骨图用于问题定义

或者是开发计划的归纳上，如图 11-41 所示。

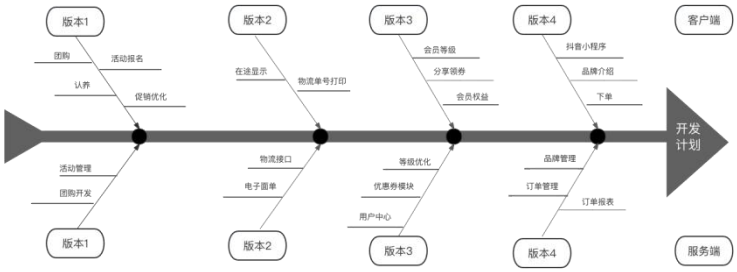

图 11-41 鱼骨图用于迭代计划

11.4.3 思维导图

思维导图又称脑图，是一种表达发散思维的图形工具，但其实真正的思维导图并非是我们平常见到的那样，如图 11-42 所示。

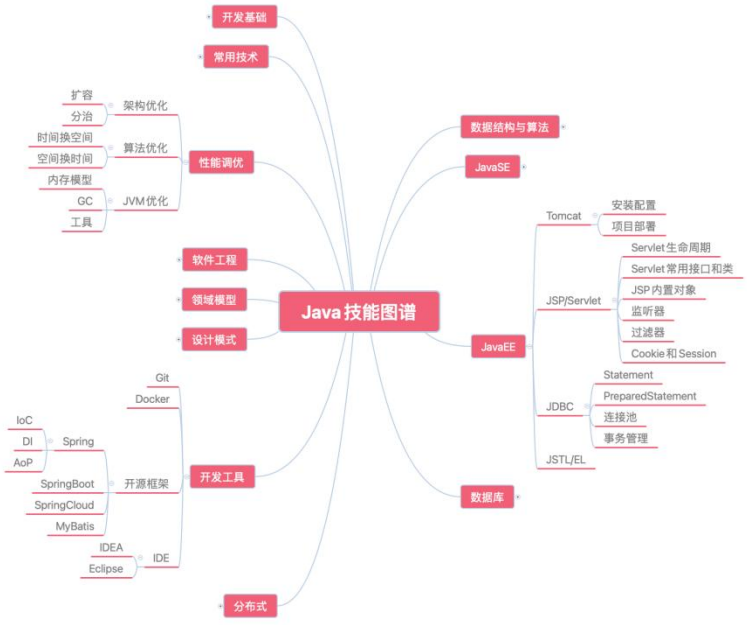

图 11-42 所谓的“思维导图”

真正的思维导图分为两种：一种是美国教育改革家 David· Hyerle（大卫·海勒）发明的 Thinking Map，另一种是 Mind Map，也就是上图看到的那种发散图，它是由英国的“记忆力之父” Tony·Buzan（托尼·布赞）发明的。Thinking Map 有八种不同的图示类型，分别对应于人在思考时的八种思维过程，而 Mind Map 仅有一种中心点向外发散的图示，如图 11-43 所示。

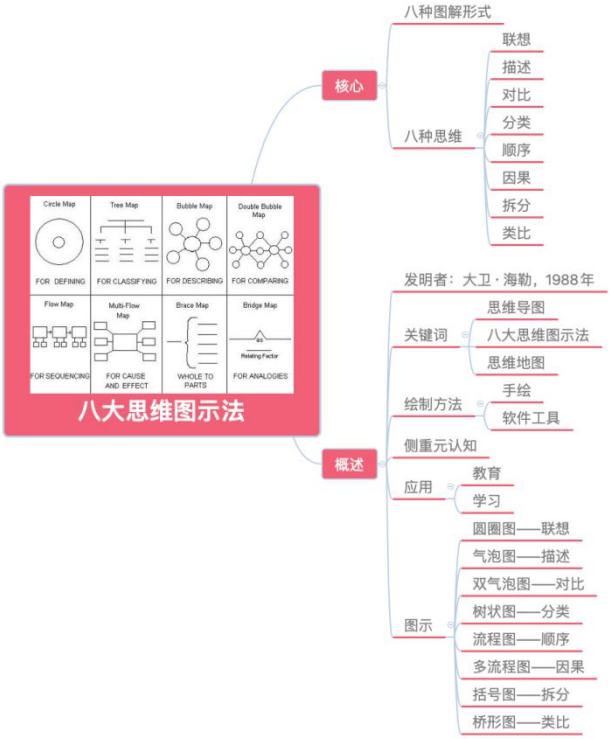

图 11-43 Thinking Map

由于流程图已经讲过，而树状图和括号图和 Mind Map 的差别不大，所以这三类就不再赘述。下面就来看看圆圈图、气泡图、双气泡图、多流程图和桥形图代表的都是什么意思。

圆圈图常用来锻炼想象力，它由一个小圆和一个大圆组成，中心词写在小圆图内，关于中心词的联想则写在两圆之间。联想出的词可以是文字，也可以是简单的图片。例如，以一次出国旅游为例，图 11-44 所示的就是圆圈图。

图 11-44 Thinking Map 圆圈图

圆圈图之所以可以锻炼想象能力，是因为它可以围绕某个点产生更有创意的想法、拓展思考问题的角度、回忆学过的知识、定义概念等，圆圈图也可以用 Mind Map 替代。

气泡图与圆圈图有些类似，它中间的大圆内写中心词，中心词周围的小圆圈内都是用来描述中心词的词汇，这些词汇一般都为形容词。图 11-45 所示的就是气泡图。

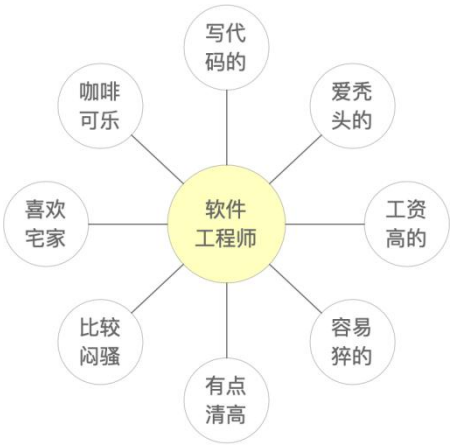

图 11-45 Thinking Map 气泡图

如果说单个气泡图是用来描述和形容某件事物或某个人的话，那么双气泡图则是用来对比两个不同的事物或人。双气泡图由两个气泡图结合而成，它有两个中心词，分别是需比较的两个事物。在两个中心词相交的气泡中是这两种事物的相同点，其他的则是不同点。图 11-46 通过双气泡图展示了项目经理与产品经理这两个关键岗位间的不同之处。

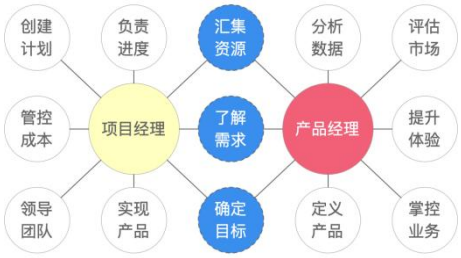

图 11-46 Thinking Map 双气泡图

多流程图可以用来确定一件事的原因和影响，它将原因与结果分列两边，让人们在头脑中自动将因果进行映射。图 11-47 通过多流程图展示了关于教育的一些问题。

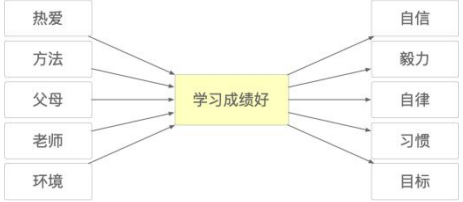

图 11-47 Thinking Map 多流程图

多流程图中的“流程”和我们所理解的“流程”是两种完全不同的东西，多流程图更像

是因果之间的一种映射关系。所以，多流程图也叫做“因果图”。

桥形图在平常工作中使用的场合并不多，它通过已知的两个事物或概念之间关系，来形容陌生的两个事物之间的关系，达到类比或隐喻的目的，如图 11-48 所示。

图 11-48 Thinking Map 桥形图图示

上图要表达的意思是：事物 1 和事物 2 之间的关系，“相当于”（或类似于）事物 A 和事物 B 之间的关系，“相当于”（或类似于）事物 α 和事物 β 之间的关系，如图 11-49 和图 11-50 所示。

图 11-49 Thinking Map 不同角色之间的关系

图 11-50 Thinking Map 不同感觉和器官之间的关系

通过桥形图，可以非常直观地让人理解“原来 A 和 B 之间的这种关系，相当于甲和乙之间的那种关系”了。

11.5 本章小节

常言道“一图胜千言”，在软件系统的设计、开发、测试和交付阶段，常用的图表包括 UML 图、ER 图、架构图和诸如甘特图、鱼骨图、思维导图等各类图表。

UML 作为建模语言，是一套专用于描述系统需求、行为、结构、交互和状态的符号规则。通过 UML 用例图，需求分析人员、项目经理或产品经理可以用一种更为精炼有效的方式，来告诉开发工程师系统需要完成什么样的功能，有哪些输入和输出，又需要遵循哪些前提条件。类图则为系统的结构和交互铺设了龙骨，描述了系统中的类、接口、关系和属性等元素，将现实世界的各种事物抽象为软件系统的虚拟实体。流程图是描述业务之间先后次序的一种图形，时序图则是从“微观”的层面来描述系统中对象与对象之间、方法与方法之间的交互行为、调用顺序和消息传递。

在数据库设计中，最常用的图表就是 ER 图，因为 ER 是着眼于实体和它们之间的关系。通过 ER 模型，工程师或系统分析员、DBA 们可以理清数据之间的关系、结构和各种约束。Navicat 提供了六种关系，它们是 None、One and Only One、Many、One or Many、Zero or One 和 Zero or Many。

从工程师成长为架构师，关键是要掌握架构思维，它们包括但不限于抽象思维、解构思

维、集成思维、分层思维、发散思维、结构化思维、迭代思维、系统思维和模式匹配等。虽说掌握这些思维、模式需要时间的沉淀，但直接套用模板也是个不错的开始。

甘特图不仅可以用于项目管理，制定个人工作计划它也是手到擒来。鱼骨图是一种类似于鱼骨形状的因果图，通过它可以定义问题，找到产生问题的根本原因和解决办法。常见的思维导图称为 Mind Map，但是其实还存在另外一种思维导图，称之为 Thinking Map，因为具有八种不同的类型的图示，因此又叫它八大思维图示法。其中的圆圈图、气泡图、双气泡图、多流程图和桥形图虽然使用场合不多，但对于想象、类比、归因等思维方式的训练，却是很好的实践方法。