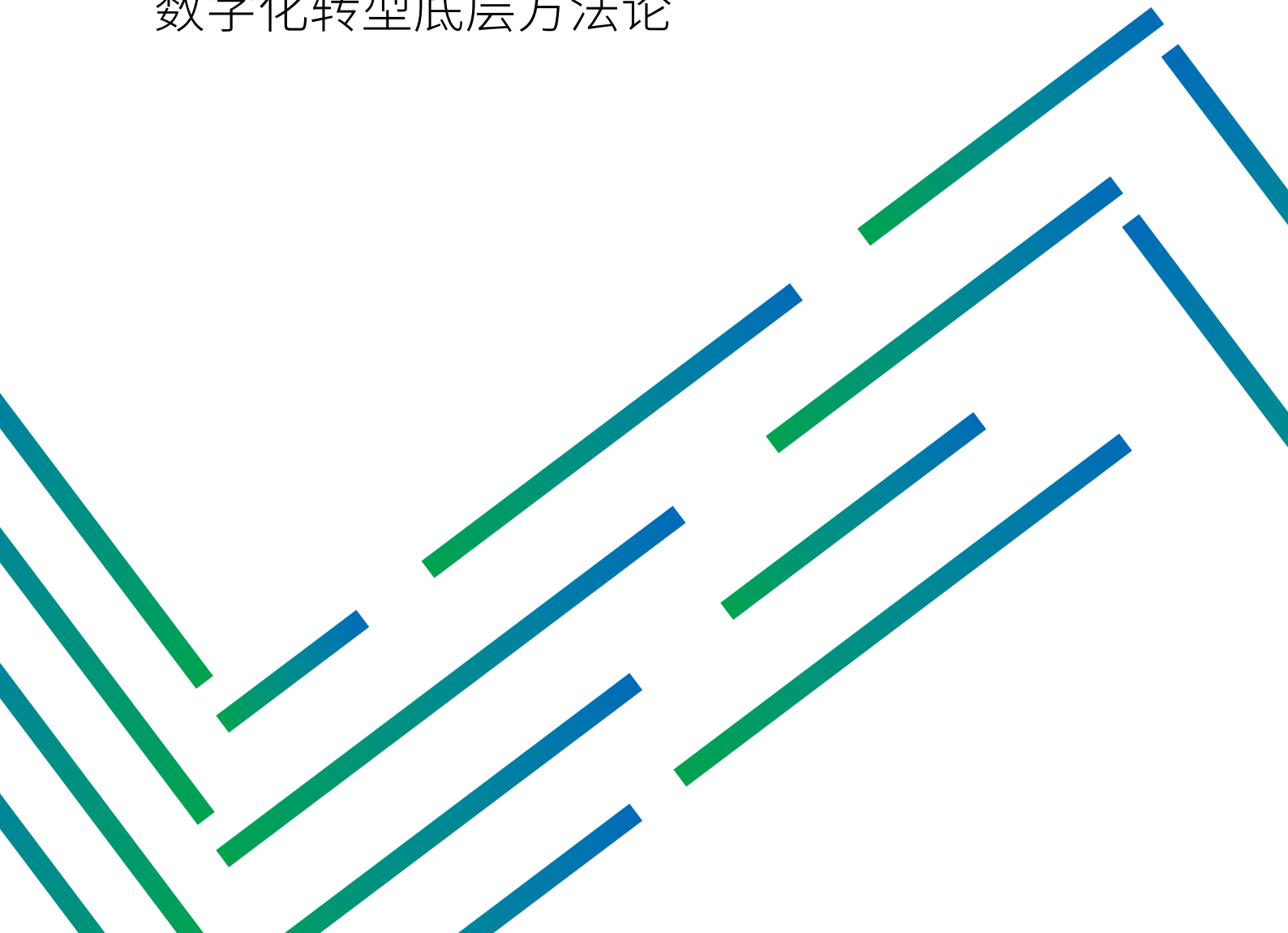
ThoughtWorks®

现代企业架构白皮书

数字化转型底层方法论



目录

1

引言：再提“业务平台化”

2020 年是“黑天鹅”集中爆发的一年，也是数字化新一波浪潮涌起的一年。数字化转型的声浪正在以加速的态势进入到各行业、各企业的战略主航道。以数字化的方式对于业务进行创新与重塑，正在成为企业新的发力点和主战场。

多项研究调查显示，60% 以上的受访高管们 (参考文献 1、2)，在疫情之后，都充分意识到数字化的必要性和对于企业带来的机遇，并表示正在着手加速企业的数字化步伐。而疫情期间，快速拥抱数字化的组织，所展现的绩效水平在其所在行业的横向对比中，也均显著优于竞争对手（参考文献 2）。

本轮以数字化驱动业务创新与重塑，越来越清晰的聚焦于：数字体验、业务平台化、智能化与云三大领域。尽管不同的行业、组织和企业，对上述领域的表述和内涵界定可能会存有差异，或仍有其他特定领域的特征显现，但在我们的观察中，这三大领域，已在跨行业范围内形成了较为广泛的共识。

本白皮书将首先聚焦于“现代企业 — 业务平台化”的趋势。我们再提“业务平台化”，源于我们在不同的行业和企业当中，反复观察和接触到、并致力于帮助解决的一系列新问题，这些问题背后所体现的趋势和解决的思路与方法，将赋予现代企业新的内涵。

1.1 “中台”概念的提出、流行到深水区实践，折射出本轮数字化转型中以“业务平台化”为代表的企业现代化趋势

1）以互联网巨头中台化为原点，延伸到以零售、金融、电信为代表的传统行业数字化建设，中台实践正在进入到深水区

从 2015 年开始，以阿里巴巴为代表的各互联网巨头，陆续开启中台化进程（如图 1）。随后，“中台”理念和相关实践开始快速向各行业渗透和发展。

零售行业

得益于阿里在零售行业的渗透和影响，以及电商在过去十年的高速发展，零售行业最先试水“中台”。一方面，阿里本身将中台理念与实践结合云服务向各行业输出，同时零售行业因业务模式的相似性，也具备较好的适配基础。另一方面，围绕电商业务模式，一系列厂商将其中标准化程度较高的部分快速“中心化”，如“商品中心”，“订单中心”，“支付中心”等，以“中台”的形态进行实施，也在一定程度上促进了中台在零售行业的推广。而零售行业的中台模型，因其本质是对于交易合同及履约的业务抽象，具备较广的适用性，也常被其他行业借鉴和扩展。

金融行业

对于金融行业，打造中台能力，无论在银行、证券或是保险等细分行业，均已是高度共识的战略举措

公司	时间	中台相关事件
阿里巴巴	2015 年 12 月	宣布全面启动“中台战略，构建符合大数据时代的“大中台、小前台”组织机制和业务机制
腾讯	2018 年 9 月	组织架构调整，“扎根消费互联网，拥抱产业互联网”，在技术上开源节流，自研上云，全面建设中台
美团	2018 年 11 月	尝试打通各个业务之间的数据，实现用户端的账户统一，实现用户数据中台
京东	2018 年 12 月	发布组织调整，采用“中台”的组织概念，中台部门已 TOB 为主，包括 3C 电子及消费品零售事业群、时尚居家平台事业群、生活服务事业群、技术中台和数据中台、商城用户体验设计部、商城市场部等六个核心部门
字节跳动	2019 年 3 月	搭建“直播大中台”，“直播大中台”将抖音、西瓜视频和火山小视频三个短视频产品抽出、合并，支撑字节跳动旗下的所有直播业务

资料来源：腾讯科技、公司公告，中银国际证券

（图 1.1-1 互联网企业的中台化大事件）（参考文献 3）

之一。（参考文献 4、5）当行业越来越重视客户一致性体验，开始打造开放银行，深度运营客户，构建生态的同时，对于长期掣肘金融行业发展的历史遗留问题，企业也深刻意识到对于此类问题解决的迫切性。如庞大的遗留系统难以响应前端快速的业务创新、客户和业务数据孤岛现象严重、组织架构壁垒高筑等。而中台建设被认为是行业普遍认同的求解之道， 根据恒生调研，半数金融机构正在考虑建设业务中台，90% 金融机构认为未来两至三年会建设业务中台。（参考文献 4、5）

电信行业

以 IT 基础设施作为重要支柱的电信行业，始终保持对 IT 新技术与工程实践的关注和引入。 过去 5 年，伴随通信技术的快速换代所带来的新的市场特征，围绕业务响应力提升和研发效能改进的目标，电信行业 IT 基础设施经历了新一轮的翻新和升级：研发体系的敏捷转型，DevOps 体系搭建、大型核心系统的容器化及服务化改造。在此基础上，加之“新基建”顶层设计的牵引和推动，行业整体的 IT 基础设施进一步深化，中台建设作为突破点和抓手之一，在营销、服务、网运等多层面展开，并不断取得进展。

2）ThoughtWorks 在以上行业中台建设的深入实践认为，“中台”不是目的，而是手段，“平台化”向业务的再演进，是本轮数字化建设中的重要趋势

回顾 ThoughtWorks 对于中台的实践，我们持续向行业输出一系列思考和洞见的同时，也广泛且深入参与了各行业的数字化建设中、尤其是以上三大行业中领跑企业的中台建设过程。

在我们的经验中，不同的行业，中台有着不同的最佳适配领域，这里从前面提到的三大行业中，我们列举一些各自领域关注的方向：

- 零售多品牌集团型企业：关注如何通过中台建设，实现集中管控前提下营销能力在多品牌复用
- 跨国零售企业：关注如何通过中台建设，实现全球统一运营下的核心业务能力针对中国市场的复用与差异化适配，以适应中国的特有业务场景和业务发展
- 商业银行与金融机构: 关注在特定业务（如信贷、资管）场景的能力抽取与模式复用，实现对于金融产品的快速创新的加速，和金融电商化的渠道能力运营的加强，为生态建设奠定基础
- 通信企业：关注如何实现跨业务线之间的公共能力的识别，沉淀，形成统一管控及支撑，实现产品平台化，更灵活的适配不同场景的需求与快速响应。

这些领域中，中台的差异化适配和建设，印证了中台实践已进入深水区。

而与此同时，中台实施“失败”的案例也不绝于耳，行业对“中台”观点，出现清晰的分化。这里，我们不展开对于中台实施失败案例的讨论与分析，而将关注点放在更加底层的商业逻辑和方法论沉淀上来。

在我们看来，中台建设的成功，或者“失败”，甚至“去中台化”声音背后，本质上是一致的商业逻辑：

中台建设“不是”广义软件套件实施：

中台以“复用”之名起源，因此，一些案例中，很容易走回从前软件套件实施的思路，以“套件化”的模式，加上一定程度的开放和定制，实现“复制”。“复制”、“复用”一字之差，却意味着从规划到落地截然不同的方法和实现。由此带来的问题是，曾经套件化实施过程中围绕“削足适履”出现过的问题、走过的弯路，在中台规划与建设中，如果仍以“中台产品套件”实施方式，会不可避免的再次出现。

中台建设“是”对于企业业务模式端到端的深度分析、建模与复用：

中台究竟在复用什么？我们给出的答案是：中台是针对于“商业模式”和“业务模式”的抽象与复用。背后体现的是企业希望通过对于自身商业模式的不断思考与认知，再通过自身业务模式的抽象与沉淀，实现跨地域、跨用户、跨场景、跨领域的扩展与复用，支撑企业业务的快速拓展与创新，也体现和契合了平台向业务的再演进过程。

借用我们一位咨询师的话：“看上去的能力复用是乐高组装，但真实的能力复用其实是器官移植，需要的是一场外科手术。”

因此，我们认同这样的观点：“中台”是手段、过程，不是目的本身。回归本源，从问题与价值出发，“平台化”向业务的再演进，是这一轮数字化建设浪潮中需要关注的重要趋势，也是企业现代化进程中的关键步骤。

1.2 再提业务平台化，是因为深水区实践中，新的问题将业务平台化内涵向前演进

“平台化”是从信息化到数字化时代，每一轮 IT 建设都会提及的主题之一。而当平台沿着历史发展的趋势继续向业务的“逼近”过程中，对于平台抽象和建设的难度也成指数型增加，涌现出了一系列新问题。

对于这些问题的思考和解决方案的探索，也将赋予业务平台化这个趋势以新的内涵和意义，同时也是我们设计和发布新的企业架构框架的起心动念。这些问题的“新”意，更多的体现在“how”上，而不再仅仅是“what”，所以我们以“如何”的句式来逐一总结和简述：

1）如何抽离多业务线共享的解决方案和能力，集中管控和演进，以避免重复投资？新业务如何基于企业已有的解决方案和能力快速组装上线，以支撑业务快速迭代创新？

今天，业务发展和 IT 建设的绑定比以往任何时间都更加紧密，而当大型企业的业务涉猎已足够广泛，多条业务线、或者多个业务单元并行发展，IT 建设会随着业务边界、组织边界，不可避免的进一步分化，也加剧了 IT 部门进行统一管控的困难。

一方面，在很多场景中，这样的分化带来了双重投资甚至多重投资的浪费；另一方面也在加剧本就存在的用户或者客户体验的割裂、数据孤岛、IT 翻新周期长等固有问题。

同时，当业务线不断尝试新的业务模式，或对于已有模式进行更快的试验、调整与扩展。对于 IT 支撑的响应力也提出了更高的要求。固有的系统搭建或者产品部署模式，越来越不足以提供及时的响应，且在“快速试错”、“小步快跑”的创新场景应对下，成本高昂。

因此，如何抽离多业务线共享的解决方案和能力，集中管控和演进，以避免重复投资？新业务如何基于企业能力快速组装上线，以支撑业务快速迭代创新？是需要解决问题。进一步拆解，我们认为需要回答：

- 针对不同的业务深度，如何设计“模式”与“能力”模型，以对业务进行合理的抽象，进而识别相似度，抽象与提炼可复用的业务模式；而针对不同业务的差异性，如何在“模式”和“能力”基础上进行扩展？
- 抽象并沉淀了业务能力之后，如何在新的业务场景中，识别、复用已有能力，应用、数据、技术及组织应该如何予以支撑？
- 以上的工作过程，是否有较好的工作实践和参考模型？

2）如何合理地划分 IT 系统边界，以得到随“需”而变的响应力

除了能力复用外，另一个提升 IT 支撑响应力的关键是，提升 IT 系统各组成部分的自治性，使得变更能够相对独立的，在更小的边界范围内，以小步快跑的方式发生，同时还需要保持一定的一致性，使整体架构做到“形散神聚”。

因为无论是创新也好，集中管控也好，虽然变化速率不同，但变化始终存在，既然变化不可避免，我们应将精力投入到应对变化上。而一个清晰的应用边界划分，可以对于业务能力通过对于知识边界的解耦，实现知识的黑盒复用，对于变化的响应非常有帮助。

在我们的经验中，应用边界划分不合理会对应对变化产生明显的负面作用。一般来说，边界的粒度过粗，容易产生功能、运行层面的变更冲突，而边界的粒度过细，则带来了额外的变更成本和性能开销，这里对各种负面作用暂不作展开，但它们的共同点是都会拖慢 IT 支撑的响应力或稳定性。

因此，“如何划分 IT 系统的应用边界，以合理的布局更好地应对变化”是需要解决的问题。进一步拆解，我们认为需要回答：

- 如何划分应用构建块的逻辑边界，使其尽可能职责单一、接口规范明确，容易变更和组装？
- 如何组合应用构建块成为合适粒度的独立可部署单元，尽可能减少功能、运行层面的变更冲突？
- 如何描述、留存以上决策的结果和依据，当变化发生时，通过溯源做出优质的新应对？

3) 如何适当拆分过于集中的分析类数据处理职责，以缓解规模化数据分析处理瓶颈？

长久以来，业界对数据架构的通用做法是：对于运行类（Operational）和分析类（Analytical）场景，应该使用不同的设计方法和技术支撑。

运行类场景以业务事务为主线，关注点在于业务事务运作证据的完整性和一致性，以及确保各类数据在各业务单元间高效、准确地传递，实现跨业务单元的事务推进以及对于业务溯源的支撑。

分析类场景则需要对内、外部数据进行收集和加工，用来度量业务运行表现、尝试分析产生偏差的原因，甚至结合机器学习等技术给出对于未来发展趋势预测和判断，尝试构建数据驱动运营的企业组织。

数据想要形成分析类价值，背后需要经过摄取（Ingest）- 加工（Process）- 能力包装（Serve）三大工序，其又可以进一步分为数据仓库、数据湖、数据与 AI 平台、数据中台等构建模式，它们都有着各自不同的适用场景和技术栈，针对这些模式的差异我们暂不作展开。

由于分析类场景所要求的方法和技术与运行类场景有着显著的不同，许多企业组建了专职的数据团队，将分析类数据处理工作和其背后的复杂性打包成为一个黑盒，提供端到端的统一的数据类企业级服务与支撑。

这个模式对于业务场景简单的企业环境工作得不错，但对于多业务线、业务平台化的企业环境已初显疲态。一方面，随着 IT 建设加速，数据源和分析类场景的数量激增，对数据类服务的响应力提出了更高要求。另一方面，想要提供高质量的数据类服务，除了分析类数据的专业技能，还要求对于业务场景、现有应用软件的深入理解。如果所有工作仍然只由专职的集中式团队一肩挑，团队带宽的限制必然会拖慢响应力。

因此，我们认为需要探索的是如何适当拆分过于集中的分析类数据处理职责，为集中式的数据团队减负，使其可以将精力投入到高价值的分析类场景中。

4）如何在富技术时代进行平台型技术架构选型及设计？

受益于云原生架构的兴起与发展，新技术的涌现和不断成熟，及技术工具的极大丰富，技术架构设计的灵活度和效率都得到了显著提升。

另一方面，在平台型技术架构的设计中，作为多业务线、多应用、多数据场景落地的技术基座，技术架构设计所需覆盖的规模、应对的复杂度今非昔比。加之“富”技术条件的加持，技术架构的设计困难度正呈现指数级增长。而一直以来，本质上是强依赖架构师的经验和能力的技术架构设计方法和过程，在这样的语境下，一系列挑战和问题再次凸显：

- 对于架构需求把握不足或者没有架构需求的分析意识，过早的进入架构设计，导致系统复杂度变高甚至过度设计，为开发落地带来额外的研发成本
- 架构设计采用的技术和工具过于超前，超出团队成员技术水平，造成落地难度高，新成员上手速度慢，进而对整体进度和实施效果造成影响
- 架构设计过程时间长，完成后团队就不再愿意对设计方案继续调整和迭代，当技术发展变化很快时，技术架构方案容易进入“完成即落伍”的困局。

因此，我们认为架构师们比以往，更需要这样一个体系：

1.3 企业架构设计方法，是有效的工作方法，经典的企业架构框架已不足够应对业务平台化中的新问题

1）以企业架构框架方法进行业务平台设计，是有效的工作方法，且各主流方法各有侧重

业内越来越普遍的采用企业架构框架，作为业务平台化整体规划指导和方法，这是有效的。因为各类企业架构框架的元模型，大体都可以归结为四类视图，即业务架构、应用架构、数据架构和技术架构，尽管不同框架在具体的层级划分、及各层结构下的内容涵盖可能会略有不同。这样的结构较好的匹配了业务平台设计的问题域。

同时各类企业架构框架的工作逻辑相似，均是从愿景与业务目标出发自顶向下的贯穿设计，并保持从业务到技术的一致性，这样的工作逻辑与业务平台从设计到落地的逻辑一致。

在此基础上，不同的企业架构框架，由于产生的背景和发展过程的不同，形成各自的侧重点或行业属性，这也从另外一个方面加强了其适用性。例如：

Zachman：侧重从利益相关者的六个视角来描述企业

TOGAF：强调企业架构全生命周期治理

DoDAF/FEAF-II：面向政府机构的投资组合管理，注重 ViewPoint

BIAN： 面向银行业，有银行业开箱即用参考模型

需要说明的是，以上的侧重总结，仅代表我们在项目操作中的理解，实际上，每一种框架在框架设计上都是完备的，在各自的领域和适用场景中也得到了广泛的应用。

2）经典的框架更注重概念的完整性，工程实操性仍显不足，且对业务平台化的新问题均没有特定的设计和考虑

下面这张表格，体系化的从概念、建模、流程的角度，对若干经典企业架构框架进行对比，从中我们可以清晰的解读出，在 Concept（概念）层面的各项评估中，各框架普遍的评定都在 H（高）和 M（中），而从 Moddling（建模）开始，到 Process（流程），评定开始从 M（中）转向 L（低），其中和落地的相关性越高的评估项，普遍的评定都位于 L（低）。

TABLE I
SUMMARY OF COMPARISON

Aspects	EAP	TOGAF	DODAF	Gartner	FEA
Concepts					
Alignment	L	M	M	M	L
Artifacts	M	H	M	M	M
Governance	M	H	M	M	L
Repository	M	M	M	M	M
Strategy	H	H	H	M	H
Modeling					
Easy to use	M	L	M	M	M
Easy to learn	M	L	M	M	M
Traceability	M	H	L	L	M
Consistency	M	H	L	L	M
Different Views	M	M	M	L	M
Complexity	L	L	L	L	L
Dynamic	L	L	L	L	L
Process					
Requirement	L	H	L	L	L
Step by Step	M	M	M	M	M
Detailed Design	M	M	M	M	M
Implementation	M	M	M	M	M
Guidelines	M	H	M	L	H
Maintenance	L	M	L	L	M
Continual	M	H	L	L	L

Notation: H: high consideration or detailed and clear description; M: medium consideration or little description; L: low consideration or high level description

（图 1.3-2 企业架构框架对比）（参考文献 6）

这张表格出自 2015 年的一篇学术文章（参考文献 6），在这之后的 5 年时间里，各框架也均不同程度地对实操的内容进行了补充和增强。但从我们的实际的跟进研究和项目经验来看，各经典框架在项目工程实操性中仍显不足。这可能也与企业级架构框架的定位相关，其大多数的定位都偏向于战略规划和组织级管理与治理，对于架构在具体设计和建模层面都没有进行细粒度展开。

同时，在第二小节中所提及的，在业务平台化的背景和趋势下我们所面临的新问题，也可映射到企业架构框架元模型的四个层次中：

- 业务架构层：如何抽离多业务线共享的能力，以避免重复投资？新业务如何基于企业能力快速组装上线，以支撑业务快速迭代创新？
- 应用架构层：在宏观规划层面，如何有效的划分和组织能力，如何划分 IT 系统的物理边界，以合理的布局更好地应对变化，在局部设计层面，如何在最大化复用效果的同时，保障对差异化需求的响应力？
- 数据架构层：如何适当拆分过于集中的分析类数据处理职责，缓解规模化瓶颈
- 技术架构层：如何在富技术时代进行平台型技术架构设计

这些在当前时代背景下广泛存在的新课题，从各经典企业架构框架中也无法直接找到针对性的设计参考和考量依据，需要我们进一步思考、实践与提炼。

1.4 ThoughtWorks 在经典企业架构框架基础上，面向以业务平台化为代表的企业现代化转型中的新问题，发布现代企业架构框架

本白皮书提出的现代企业架构框架就是在这样的背景下，针对以业务平台化为代表的企业现代化转型过程中的背景及挑战，从实践中探索和总结出的一套轻量级、敏捷、可落地的企业架构框架方法。

整体框架方法设计保持对经典企业架构框架优秀设计部分的传承，从框架元模型整体结构上，仍遵循经典企业架构框架的最佳实践，延续基于业务架构、应用架构、数据架构及技术架构的架构视图分类。而针对业务平台化的特征及新问题，对各层元模型内容，进行了扩展和再定义。

与此同时，这套框架力求实操，对于每一层模型的讲解，我们都以问题作为牵引，以解决问题的实操过程作为内容串联主线。因此，每一个部分的详细介绍中，我们按照这样的顺序进行讲解，先给出该层元模型概览，之后回顾问题，以问题的解决过程作为牵引，讲解元模型的应用。

2

现代企业架构框架（Modern Enterprise Architecture Framework – MEAF）

企业架构（Enterprise Architecture）始于 20 世纪 60 年代，截至目前已有接近六十年的发展历程，作为一门关键的 IT 学科领域，经过多年的发展也催生了各类广泛应用于各行业和应用场景的框架与方法论工具，例如 Zachman、TOGAF、DoDAF 等，这些企业架构框架也一直作为重要的指导方法和工具，被应用于各类企业和组织的顶层 IT 规划与设计。

但随着近些年互联网的高速发展，“互联网速度”和“产品为王”的理念直入人心，也导致企业将更多注意力和资源聚焦于产品（系统）架构层面，关注于如何通过合理的系统架构设计帮助产品快速抢占市场和用户，获得成功。

这样的产品驱动型策略，切实帮助了很多企业快速通过核心产品的构建和发布，迅速占领市场，获得了阶段性成功。但同样也是因为缺乏对于企业级架构的整体规划与设计，当企业逐渐深入到可持续发展和业务持续拓展阶段，产品重复建设、技术与架构大规模异构等问题逐渐显现，给企业带来了越来越大的负担，拖慢了企业持续发展和持续创新的速度。

因此，企业架构重新被大家关注和重视，以互联网巨头为代表开始了一系列的企业级架构重新规划和治理的工作，其中“中台战略”就是典型的代表。这样的趋势也正在从互联网行业蔓延到其他行业，掀起了一波企业级架构规划和治理的新浪潮。

至此，企业架构和企业架构框架又重新回归大家的视野，成为企业架构治理和企业平台化转型，乃至企业数字化转型的重要理论依据和指导工具。

2.1 企业架构

在展开描述企业架构和企业架构框架之前，首先追根溯源，了解一下架构的含义，在 ISO/IEC/IEEE-42010:2011 标准中对于架构的定义是：

> **架构是系统在其所处环境中的基本概念或属性，体现为它的元素、关系，以及系统设计和演进的原则。**

架构这个概念源于建筑等其他行业，随着计算机行业的兴起，这样的概念也被引入到了信息技术行业，用于 IT 系统的设计。在信息技术领域大体上主要分为两个粒度，即：系统架构（System Architecture）与企业架构（Enterprise Architecture）。

信息技术领域的架构设计本质是一个认知、抽象与构建的过程，即通过对于物理世界的认知与抽象，识别其中的关键概念及其关系，再通过数字化的手段在数字化世界里重新构建、模拟和还原。

而企业架构同样作为一种架构体系，也依然符合对于架构的概念定义，只是将关注点从系统级别提升到了企业级别，即企业架构关注的是在企业级别的各种视角（viewpoint）及其视图（view），其中的基本元素及其关系。

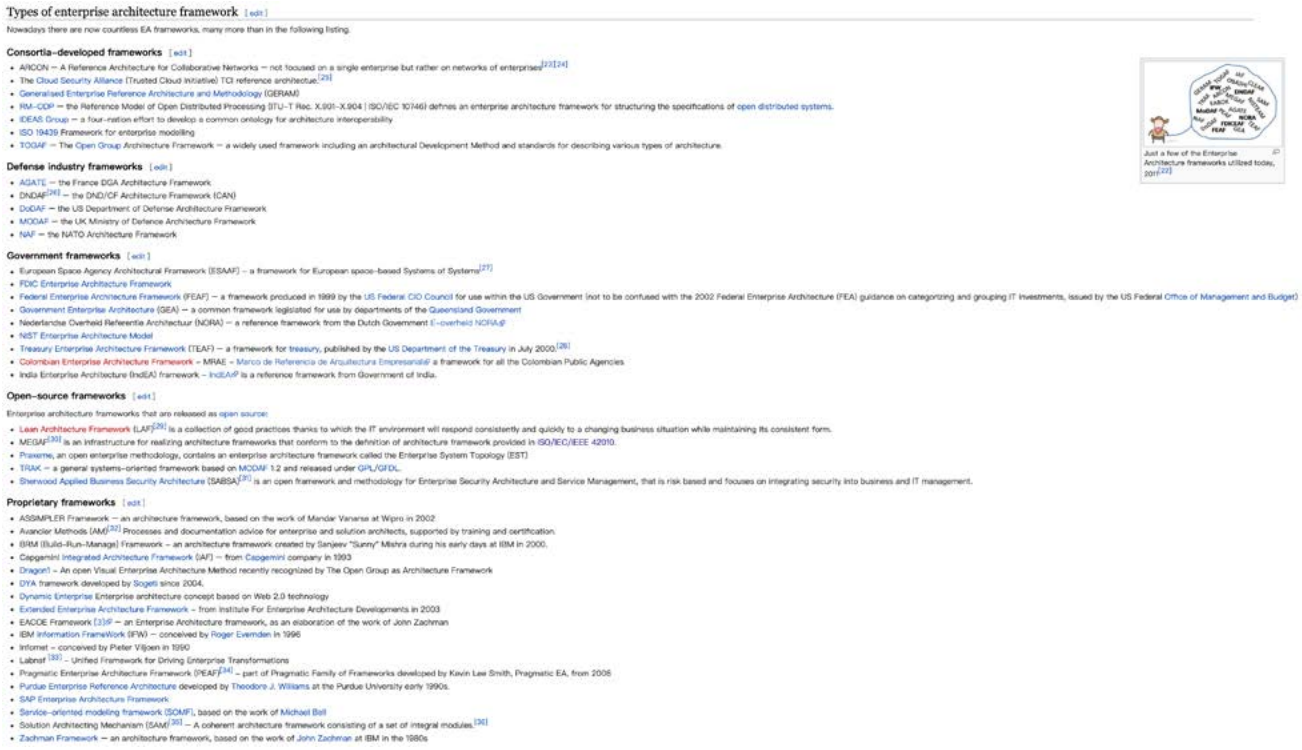
通过对于企业架构的规划和设计，可以帮助企业构建整体的数字化策略，规划数字化项目，通过数字化的手段帮助其实现期望的战略目标和业务结果，形成企业的数字化顶层规划与设计，指导企业的数字化转型过程。而这个企业架构规划的过程，也被称为企业架构规划 (enterprise architecture planning, EAP)。

2.2 企业架构框架

很多人将企业架构（Enterprise Architecture）与企业架构框架（Enterprise Architecture Framework）的概念混淆，就像很多人容易将敏捷（Agile）与轻量级软件开发方法（如Scrum）混淆一样，其实这是两个完全不同的概念。

企业架构是一门领域学科，而企业架构框架（例如常见的 TOGAF）才是一种具体可实施的框架和方法论，企业架构与企业架构框架的关系，就类似于敏捷（Agile）与轻量级软件开发方法（例如 Scrum、XP）的关系一样。

在众多的企业架构框架方法之中，最被大家熟知通用型企业架构框架当属 TOGAF，其已经成为通用行业企业架构框架的标准方法。而近些年大量新涌现的轻量级企业架构方法，也大多从 TOGAF 发展而来，或是对于 TOGAF 进行扩展，或是对于 TOGAF 进行细化和补充。



Types of enterprise architecture framework [edit]

Nowadays there are now countless EA frameworks, many more than in the following listing.

Consortia–developed frameworks [edit]

- ARCON – A Reference Architecture for Collaborative Networks – not focused on a single enterprise but rather on networks of enterprises[23][24]
- The Cloud Security Alliance (Trusted Cloud Initiative) TCI reference architecture[25]
- Generalised Enterprise Reference Architecture and Methodology (GERAM)
- RM–ODP – the Reference Model of Open Distributed Processing (ITU–T Rec. X.901–X.904 | ISO/IEC 10746) defines an enterprise architecture framework for structuring the specifications of open distributed systems.
- IDEAS Group – a four–nation effort to develop a common ontology for architecture interoperability
- ISO 19439 Framework for enterprise modelling
- TOGAF – The Open Group Architecture Framework – a widely used framework including an architectural Development Method and standards for describing various types of architecture.

Defense industry frameworks [edit]

- AGATE – the France DGA Architecture Framework
- DNDAF[26] – the DND/CF Architecture Framework (CAN)
- DoDAF – the US Department of Defense Architecture Framework
- MODAF – the UK Ministry of Defence Architecture Framework
- NAF – the NATO Architecture Framework

Government frameworks [edit]

- European Space Agency Architectural Framework (ESAAF) – a framework for European space–based Systems of Systems[27]
- FDIC Enterprise Architecture Framework
- Federal Enterprise Architecture Framework (FEAF) – a framework produced in 1999 by the US Federal CIO Council for use within the US Government (not to be confused with the 2002 Federal Enterprise Architecture (FEA) guidance on categorizing and grouping IT investments, issued by the US Federal Office of Management and Budget)
- Government Enterprise Architecture (GEA) – a common framework legislated for use by departments of the Queensland Government
- Nederlandse Overheid Referentie Architectuur (NORA) – a reference framework from the Dutch Government (i–overheid NORA)
- NIST Enterprise Architecture Model
- Treasury Enterprise Architecture Framework (TEAF) – a framework for treasury, published by the US Department of the Treasury in July 2000.[28]
- Colombian Enterprise Architecture Framework – MRAE – Marco de Referencia de Arquitectura Empresarial a framework for all the Colombian Public Agencies
- India Enterprise Architecture (IndEA) framework – IndEA is a reference framework from Government of India.

Open–source frameworks [edit]

Enterprise architecture frameworks that are released as open source:

- Lean Architecture Framework (LAF)[29] is a collection of good practices thanks to which the IT environment will respond consistently and quickly to a changing business situation while maintaining its consistent form.
- MEGAF[30] is an infrastructure for realizing architecture frameworks that conform to the definition of architecture framework provided in ISO/IEC/IEEE 42010.
- Praxeme, an open enterprise methodology, contains an enterprise architecture framework called the Enterprise System Topology (EST)
- TRAK – a general systems–oriented framework based on MODAF 1.2 and released under GPL/GFDL.
- Sherwood Applied Business Security Architecture (SABSA)[31] is an open framework and methodology for Enterprise Security Architecture and Service Management, that is risk based and focuses on integrating security into business and IT management.

Proprietary frameworks [edit]

- ASSIMPLER Framework – an architecture framework, based on the work of Mandar Vanarse at Wipro in 2002
- Avancier Methods (AM)[32] Processes and documentation advice for enterprise and solution architects, supported by training and certification.
- BRM (Build–Run–Manage) Framework – an architecture framework created by Sanjeev "Sunny" Mishra during his early days at IBM in 2000.
- Capgemini Integrated Architecture Framework (IAF) – from Capgemini company in 1993
- Dragon1 – An open Visual Enterprise Architecture Method recently recognized by The Open Group as Architecture Framework
- DYA framework developed by Sogeti since 2004.
- Dynamic Enterprise Enterprise architecture concept based on Web 2.0 technology
- Extended Enterprise Architecture Framework – from Institute For Enterprise Architecture Developments in 2003
- EAGDE Framework [33]? – an Enterprise Architecture framework, as an elaboration of the work of John Zachman
- IBM Information FrameWork (IFW) – conceived by Roger Evernden in 1996
- Informet – conceived by Pieter Viljoen in 1990
- Labnet[33] – Unified Framework for Driving Enterprise Transformations
- Pragmatic Enterprise Architecture Framework (PEAF)[34] – part of Pragmatic Family of Frameworks developed by Kevin Lee Smith, Pragmatic EA, from 2006
- Purdue Enterprise Reference Architecture developed by Theodore J. Williams at the Purdue University early 1990s.
- SAP Enterprise Architecture Framework
- Service–oriented modeling framework (SOMF), based on the work of Michael Bell
- Solution Architecting Mechanism (SAM)[35] – A coherent architecture framework consisting of a set of integral modules.[36]
- Zachman Framework – an architecture framework, based on the work of John Zachman at IBM in the 1980s

(图 2.2-1 企业架构框架列表) (参考文献 10)

2.3 现代企业架构框架

虽然如上所述，像 TOGAF 这样的经典企业架构框架从诞生到今已经历了 20 多年的发展，在发展和演进过程中也与时俱进地加入了对于像 SOA 等新的架构模式的支持。但在具体应用框架方法实践与解决现在化企业所

面临的问题时，例如如何在云与分布式时代，基于平台思维进行企业架构设计的过程中仍显繁重且不完全匹配。

终究其原因，这类经典企业架构框架所诞生的时代仍处于信息化时代的早期，设计之初主要面对和解决的是企业信息化的问题，虽然也在持续保持演进和发展，但大多以补丁（例如通过元模型扩展的方式）的方式予以支持，并不能做到与最新的企业发展理念和技术趋势无缝融合与原生支持，尤其是在以平台型为主要特征的现代企业架构的规划与设计过程中，最典型的就是国内目前比较广泛采用的中台架构规划过程中，略显乏力。

因此，不破不立，能否在充分吸收经典企业架构框架的优秀思想和最佳实践的前提下，融合最新的企业数字化发展的需求和新技术新趋势，勇于跳出 TOGAF 的限制，从问题出发，回归第一性，重新思考和构建一个新的轻量级企业架构框架，切实可以解决企业在向现代企业架构演进过程中面临的问题和挑战，就成为我们重点关注和研究的领域。通过几年的研究实践，也逐渐形成了一套轻量化、敏捷可落地的企业架构框架方法，我们把它定义为：现代企业架构框架（Modern Enterprise Architecture Framework）。

2.4 现代企业架构框架设计原则

当我们在总结和提炼现代企业架构框架（MEAF）时，为了保证框架设计的有效和易于实施，从框架设计之初就一直遵循以下框架设计原则：

- **战略与业务价值驱动（业务驱动 over 技术驱动）**

战略与业务价值驱动，是框架设计的第一个重要原则，无论是框架本身的设计还是应用框架进行企业级的架构规划，都需要始终遵循此规则，使每一个架构决策都能回溯到企业的战略方向和业务价值上。

为了达到此目标，无论是企业级的应用架构还是企业级的技术架构，都需要以支撑企业级的业务架构为目标，而企业级的业务架构要能直接对应和反应企业的战略方向以及业务价值体现。

一切从业务出发，以价值驱动，是架构设计的重要原则与基础。

- **轻量敏捷化（持续改进 over 一次做对）**

为保证架构的轻量，从框架设计之初，团队一直反复审视框架每一个概念和工具的价值和成本，在满

足现代企业架构设计的前提下，力求用最少的概念和元素解决实际问题。

同时如何让企业架构与敏捷的思想融合，也是我们在框架设计之初就一直探讨和研究的课题。我们希望同时结合 ThoughtWorks 在敏捷与企业架构及平台架构各个领域的优势和理解，使现代企业架构框架原生就融入敏捷的思想、原则和最佳实践，扭转大家对于企业架构“繁重、复杂、成本高、不落地”的固有认识。

为达到此目标，我们将数据驱动、持续迭代、小步快跑、协同共创、工作坊等思想、方法和工具也融入到了现代企业架构框架之中，帮助在框架实施过程中实现企业级架构规划与建设的敏捷化。

- **可落地（从实践出发 over 从理论推导）**

本框架在设计过程中的所有元模型定义和方法建议，都源于实际项目的实践和提炼，因为可落地易落地也一直是我们构建和设计这个框架的重要原则之一，任何好的概念、思想和工具，如果没有经过实践的检验，也不会被加入到框架的核心模型和要素中来。

反映到框架设计上，从框架的核心元模型出发，每一个元模型要素都会包含完整的概念定义、应用场景、建模语言标准、识别方法与工具建议、输入基线要求、输出基线定义，以确保框架的可落地和应用此框架设计与建模的一致性，同时降低框架掌握的门槛，做到易懂、易学、易用。

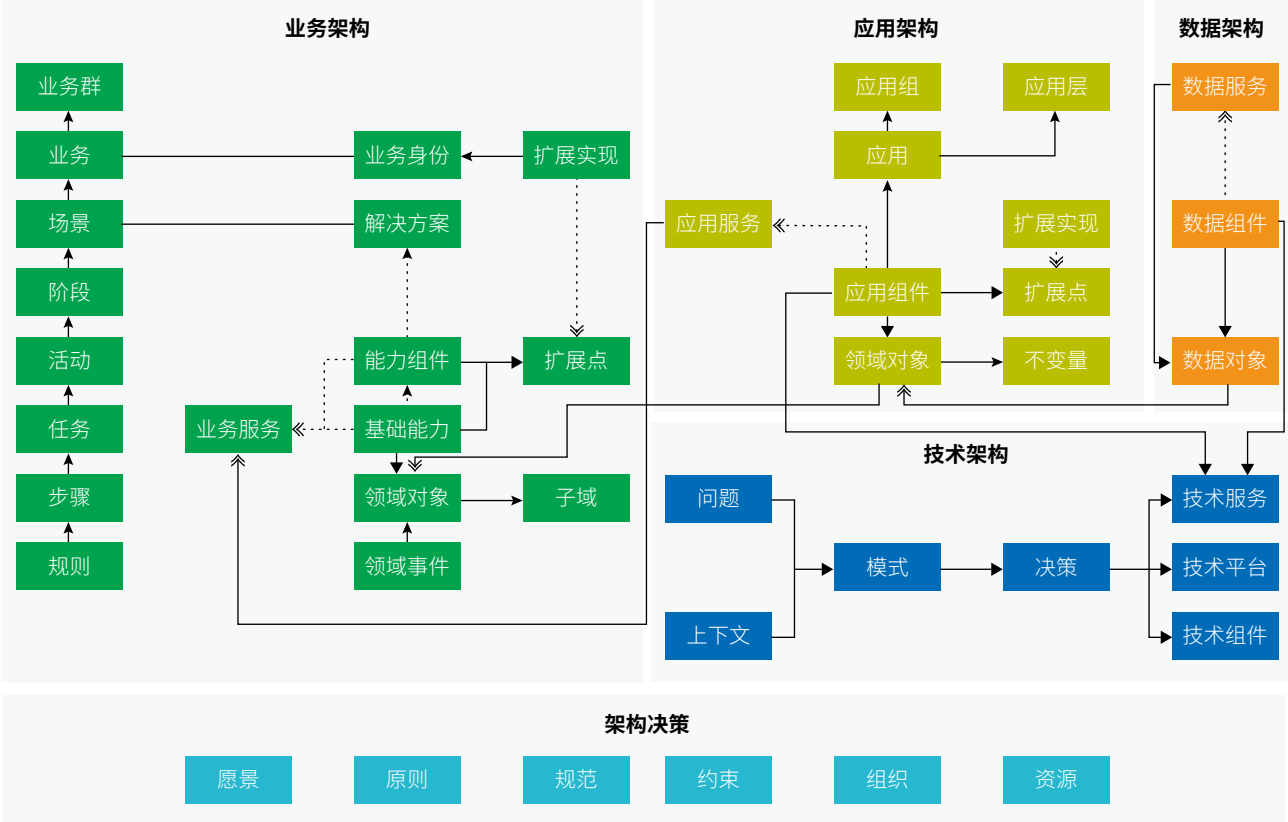
同时，对于框架从元模型出发到设计与建模方法，支持基于企业的自身特点和不同的战略目标以及业务要求，支持对于框架做出适当的裁剪、扩展和定制，使框架切实成为企业级架构规划的有力支撑而非固化限制。

2.5 现代企业架构框架元模型总览

在展开介绍架构之前，我们需要先了解在企业架构领域非常重要的三个概念：元模型（Metamodel），视角（viewpoint）和视图（View）：

元模型（Metamodel）：元模型是对于架构核心概念要素的精确定义和描述，元模型构成了架构设计的“基本语言要素”，通过元模型及其关系的表达，就可以通过结构化的方式对于架构进行描述和展现，框架元模型体现了框架设计者对于企业级架构本身的理解和抽象，是企业级架构框架的核心，是对于架构描述的“统一语言”。

视角（Viewpoint）：企业架构设计因为是在对于企业本身的进行架构设计，因其抽象程度较高，同



(图 2.5-2 MEAF Metamodel)

时涉及各类不同的干系人和组织，而不同的干系人和组织基于自身所处岗位角色和职责的不同，对于架构的关注点和视角也存在比较大的差异。因此，通过不同的视角（Viewpoint）的抽象，就可以充分体现我们在审视和进行企业架构设计时，处于什么样的观察位置和角度，兼顾不同干系人的架构设计诉求。不同的视角（Viewpoint）会关注架构的不同切面，以及在这个切面下的元模型要素以及他们之间的关系，这就构成了不同的架构视图（View）。

视图（View）：一个视图描述了从一个或一组相关的视角（Viewpoint）出发，通过组合这类视角所关注的元模型（Metamodel）要素及其关系，通过设计与建模之后，形成的切面视图。一个视图（View）体现了在一类视角（Viewpoint）下对其关注的架构元模型要素及其关系的描述和可视化。

在现代企业架构框架（MEAF）的设计上，我们最大化的延续和集成了经典企业架构框架对于视角（Viewpoint）和视图（View）的划分，当前版本主要从业务架构、应用架构、数据架构和技术架构出发四类架构视图出发，将关注点聚焦于在不同架构视图下，针对平台型企业架构设计这个大的前提和背景，如何设计和应用元模型（Metamodel）重新对于企业架构建模，满足企业对于现代企业架构设计的需求的同时，保证企业架构设计的可落地。

下面就将根据不同的视图（View）以元模型（Metamodel）为主线，展开详细介绍现代企业架构框架（MEAF）的架构设计要素和应用场景。

3

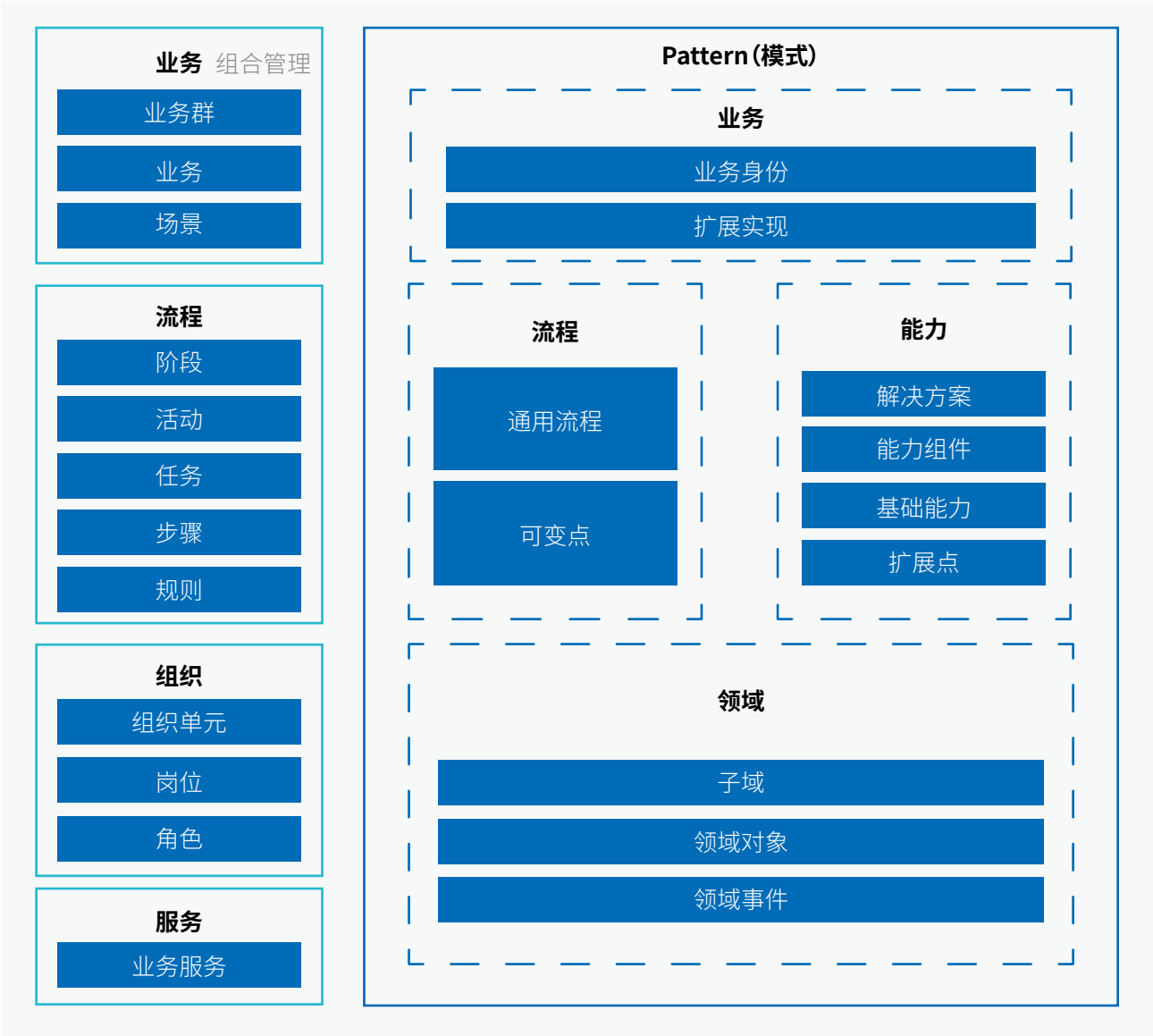
现代企业架构框架 — 业务架构

3.1 业务架构元模型综述

业务架构 (Business Architecture) 定义了企业各类业务的运作模式及业务之间的关系结构。它以承接企业战略为出发点，以支撑实现企业战略为目标，通过对于业务能力的识别与构建，并将业务能力以业务服务的方式透出，实现对于业务流程的支撑，并最终通过组织给予保障。

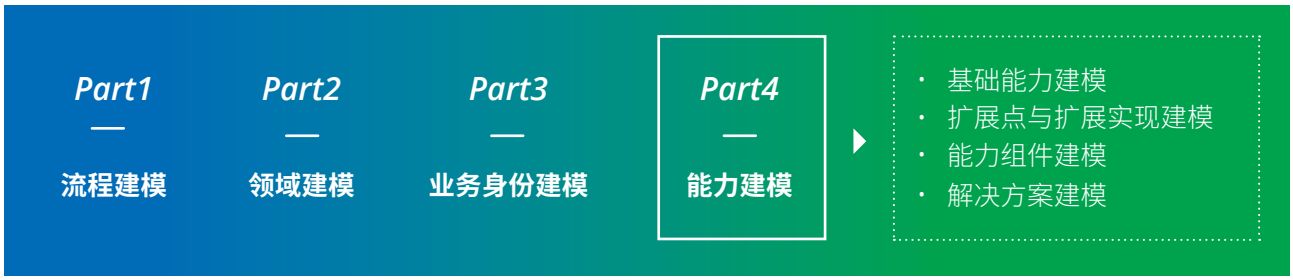
业务架构是企业架构的核心内容，直接决定了企业战略的实现能力，是其他架构领域工作的前提条件和架构设计的主要依据。

业务架构整体上包括“业务”、“流程”、“组织”、“服务”、“领域”和“模式”六大部分，如下图 3.1-1 所示：



（图 3.1-1 企业级业务架构元模型）

其中“模式”部分是我们为“平台型”企业架构设计的核心解决方案，包括：



3.2 业务架构元模型应用

3.2.1 现代业务架构典型问题

在帮助企业构建业务架构的过程中，我们发现大部分企业正面临共同的问题：如何抽离多业务线共享的能力，集中管控和演进，以避免重复投资？新业务如何基于企业能力快速组装上线，以支撑业务快速迭代创新？

问题的背景和起因在于，当大型企业的业务发展到达一定规模，多条业务线并存、或多个业务单元并行发展，IT 建设会随着业务边界、组织边界，不可避免的进一步分化，也加剧了 IT 部门进行统一管控的困难。

一方面，在很多场景中，这样的分化带来了双重投资甚至多重投资的浪费，另一方面也在加剧本就存在的用户或者客户体验的割裂、数据孤岛、IT 翻新周期长等固有问题。

同时，当业务线不断尝试新的业务模式，或对于已有模式进行更快的试验、调整与扩展。对于 IT 支撑的响应力也提出了更高的要求。固有的系统搭建或者产品部署模式，越来越不足以提供及时的响应，且在“快速试错”、“小步快跑”的创新场景应对下，成本高昂。

为了解决上述问题，我们对问题进行了进一步拆解：

· Q1：什么是可共享复用的能力？

· Q2：如何识别和构建能力？

· Q3：如何使用能力，实现新业务快速上线？

在对问题进行上述拆解后，我们将基于业务架构元模型逐一解决。

3.2.2 什么是可共享复用的能力？

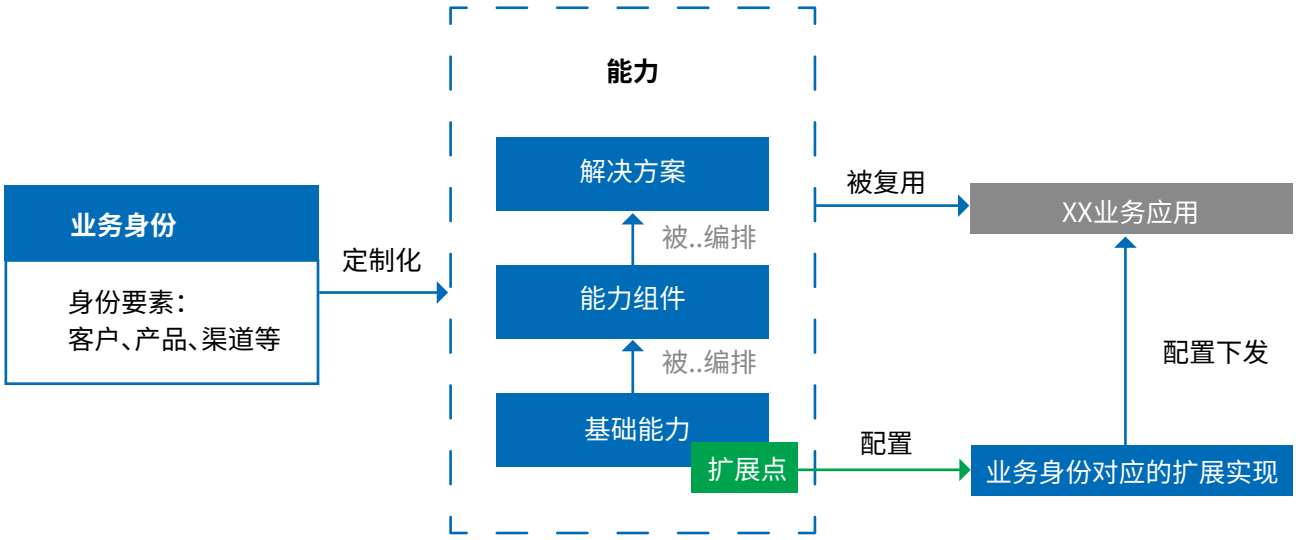
在现代企业架构中，面向能力的规划超越面向功能与服务的规划成为企业级业务架构规划的关注要点，如何基于能力的识别与规划，最大化的沉淀企业级可复用的能力，并通过扩展、编排和组合等形式应用到更多的场景，是平台型企业架构需要解决的关键问题。

企业为了应对业务的快速迭代、多场景和不确定性，需要在平台上构建可复用的“能力”以及为能力提供必要的扩展与可变机制，以此为不同前台提供灵活多变的业务服务，满足不同前台差异化个性化的的需求。

而“能力”根据粒度的不同，可再度细分为“基础能力”、“能力组件”和“解决方案”三个层级。

不同业务的差异性，则可通过能力的“扩展点”设计和不同“业务身份”在扩展点上的“扩展实现”进行区分。

总体实现机制如下：



（图 3.2-2 企业能力共享复用的实现机制）

业务身份：“业务身份”的概念最早由阿里巴巴提出，业务平台在对各业务同时提供服务时，需要能区分每一次业务服务请求的业务身份要素，以便提供差异化个性化的服务；因此需要对企业各业务的身份和特征进行建模和区分，其产出即为“业务身份”。业务身份是业务在平台中的代名词，是在业务运营中唯一区分某个具体业务的 ID。平台基于业务身份匹配该特定业务的流程和业务规则，并基于业务身份实现服务路由、需求溯源、业务监控和业务隔离。

基础能力：是对领域对象的原子操作，完成一个领域对象上单一且完整的职责。比如：创建售后单、修改商品库存量等，是能力组合和复用的最小单元。

能力组件：能力组件是对基础能力的进一步封装，目的是方便业务的使用。按封装粒度不同分为两类：第一类能力组件是根据业务服务的需要编排封装的一组关联的基础能力，从而提供完整的服务。比如：订单创建能力组件。第二类能力组件是平台针对一系列紧密关联的业务活动，设计的能力模板，可基于该模板快速定制某个具体业务的特定流程和能力，从而达到复用全部关联能力的目的。比如：“组合支付”、“快速建站”等能力组件。能力组件加快了业务接入平台的速度，让业务侧专注业务本身，不再需要耗费精力在理解平台大量的基础能力上。

扩展点与扩展实现：“扩展点”是对基础能力的可变性设计，在技术侧体现为基础能力实现中的某一个步骤的接口定义，而接口的一个实现即为一个“扩展实现”。比如：订单渲染基础能力中有一个步骤是订单总价试算，而正常时期的总价试算规则与秒杀时期的总价试算规则是不同的，因此需要对订单渲染基础能力设计“订单总价试算规则”的扩展点，并分别定义在正常时期和秒杀时期的扩展实现。

解决方案：是平台针对一类共性业务的端到端过程设计的能力模板；可基于该模板快速定制某个具体业务的特定能力和流程，从而达到业务模式级别复用的目的。比如：虚拟物品交易解决方案。

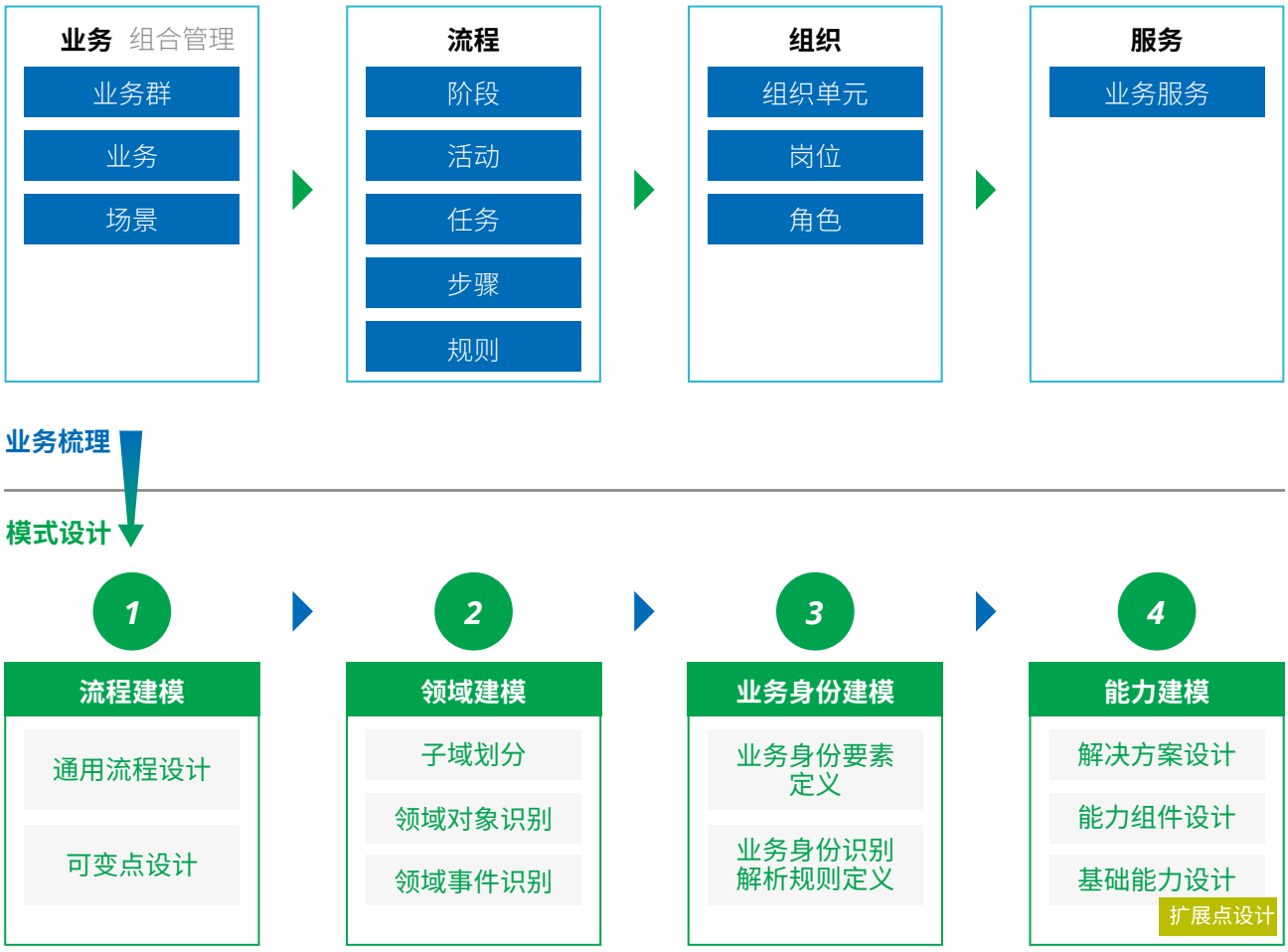
3.2.3 如何识别和构建能力?

识别构建能力的过程分为“业务梳理”和“模式设计”两个阶段。

在业务梳理阶段：对企业业务、流程、组织、业务服务和业务规则进行细致完整地梳理，作为后续模式设计的基础和输入。

而在模式设计阶段：则会通过流程建模、领域建模、业务身份建模和能力建模 4 个步骤完成企业能力的识别构建。

具体过程如下：



(图 3.2-3 识别和构建能力的过程)

模式设计阶段中：

- 流程建模：负责识别共性业务，并抽取通用流程，设计可变点，作为可复用性分析的基础。
- 领域建模：负责基于流程建模结果，识别领域事件和领域对象，并划分子域的边界；领域对象构成了提供可复用能力的基本单元，对领域对象的操作即是基础能力。
- 业务身份建模：负责定义业务身份识别的要素和业务身份解析规则，用于给可复用的能力区分不同的业务身份要素。
- 能力建模：负责最终完成平台三类可复用能力的设计，即“基础能力”设计、“能力组件”设计和“解决方案”设计。

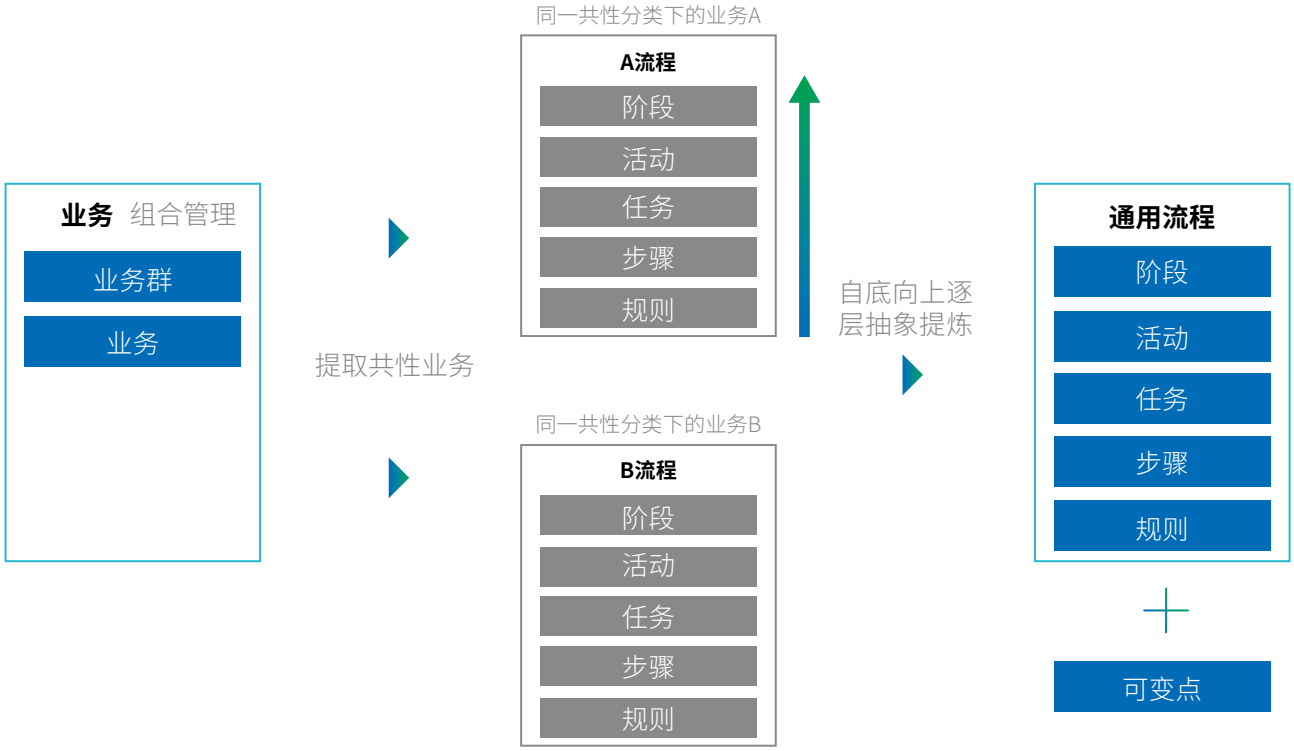
下面将详细说明各步骤的实施方法：

3.2.3.1 流程建模

为了提取可复用的能力，首先需要识别共性业务，然后将同一类共性的业务抽象提炼出通用流程，并基于通用流程进行可变性分析。

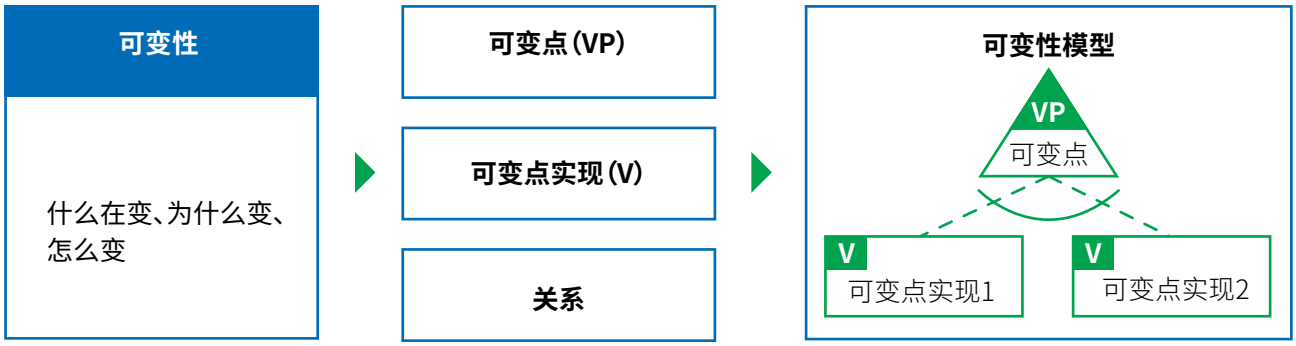
具体方法是按业务架构中流程部分的元模型（阶段、活动、任务、步骤、业务规则），结合自上而下以及自下而上的方式逐层提炼收敛通用模型和可变点。

总体实现机制如下：



(图 3.2-4 流程建模的总体实现机制)

提炼通用流程后的可变性分析，旨在找出“什么在变”、“为什么变”和“怎么变”，因此可变性模型主要由：“可变点”、“可变点实现”以及“可变点、可变点实现之间的关系”三部分组成。



（图 3.2-5 可变性模型的组成部分）

综上所述，流程建模的主要步骤如下：

- 分析业务组合，提取共性业务。
- 分析所有共性业务的各流程步骤及其输出对象，抽象提炼通用步骤和业务实体；识别各业务的差异部分，提炼设计可变点，确定可变点实现和关系。
- 分析所有共性业务的各流程任务，抽象提炼通用任务；识别各业务在任务上的差异部分。
- 分析所有共性业务的各流程活动，抽象提炼通用活动；识别各业务在活动上的差异部分。
- 分析所有共性业务的各流程阶段，抽象提炼通用阶段；识别各业务在阶段上的差异部分。

3.2.3.2 领域建模

领域是指组织的业务范围以及在其中所进行的活动，也就是平台能力需要支撑的业务范围。因此，为了构建出平台能力，需要对业务领域进行深入的理解和设计。

在业务架构部分，将进行领域战略层级的建模，主要包括：“子域”、“领域对象”、“领域事件”部分的设计。

领域
子域
领域对象
领域事件

（图 3.2-6 领域战略层级建模的组成部分）

领域战略层级建模的过程如下：

(图 3.2-7 领域战略层级建模的过程)

3.2.3.2.1 领域事件识别

领域事件（Domain Event）：是领域专家关心的，在业务上真实发生的事件，这些事件对系统会产生重要的影响，如果没有这些事件的发生，整个业务逻辑和系统实现就不能成立。我们可以通过领域事件对过去发生的事情进行溯源，因为过去所发生的对业务有意义的信息都会通过某种形式保存下来。比如：“用户地址已更新”、“订单已发货” 等领域事件。

领域事件对系统常见的影响有：

- 对内
  - 产生了某种数据
  - 触发了某种流程或事情
  - 状态发生了某种变化
- 对外
  - 发送了某些消息

目前比较常用的领域事件识别方法是“事件风暴（Event Storming）”，主要步骤如下：

- 邀请业务专家（或领域专家）和技术专家共同参与事件风暴工作坊，其它参与角色按需补充。
- 明确和选择需要分析的业务场景。
- 确定起始事件和结束事件，事件以“XXX 已 YYY”的形式进行命名（对于英文版过去完成时的中文表达方法）。
- 根据场景和业务复述的复杂度，决定以时间线的哪个方向开始梳理事件（正向或逆向）。
- 以先发散再收敛的方式，按照时间线的先后顺序和并行关系，补充和完善领域事件。
- 使用“规则”抽象分支条件或复杂的规则细节，通过抽象降低分支复杂度，规则以“XXX 规则”的名词形式进行命名。
- 完成一遍事件梳理之后，通过问问题的方式，逆向检查（Reverse Check）事件流的逻辑合理性，例如：
  - 该事件真的需要在系统实现的时候考虑吗？
  - 该事件如果存在，那它的前提条件是什么？
  - 该事件如果要产生，那它的前一个事件必须是？
- 重复以上步骤，迭代式的完成全部领域事件的识别。

领域事件识别的示例如下：

(图 3.2-8 领域事件识别的示例)

3.2.3.2.2 领域对象识别

领域对象（Domain Object）：是对业务的高度抽象，作为业务和系统实现的核心联系，领域对象封装和承载了业务逻辑，是系统设计的基础。

领域建模中重要的部分之一就是对“领域对象”及领域对象之间关系的识别和设计。而领域对象识别将基于前面领域事件识别的结果开展。

领域对象，通常包含（但不限于）：

- 领域事件中出现了的名词；
- 如果没有信息系统，在现实中会看得见摸得着的事物（例如订单）；
- 虽然在当前业务中看不见摸不着，但是可以在未来抽象出来的业务概念。

在领域驱动设计（Domain-Driven Design）（参考文献 7）中一般存在三类领域对象：

聚合根：是领域对象的根节点，具有全局标识，对象其它的实体只能通过聚合根来导航；如订单可以分为订单头和订单行，订单头是聚合根，它包含了订单基本信息；订单行是实体，它包含订单的明细信息，聚合跟所代表的聚合实现了对于业务一致性的保障，是业务一致性的边界。

实体：是领域对象的主干，具有唯一标识和生命周期，可以通过标识判断相等性，并且是可变的，如常见的用户实体、订单实体；

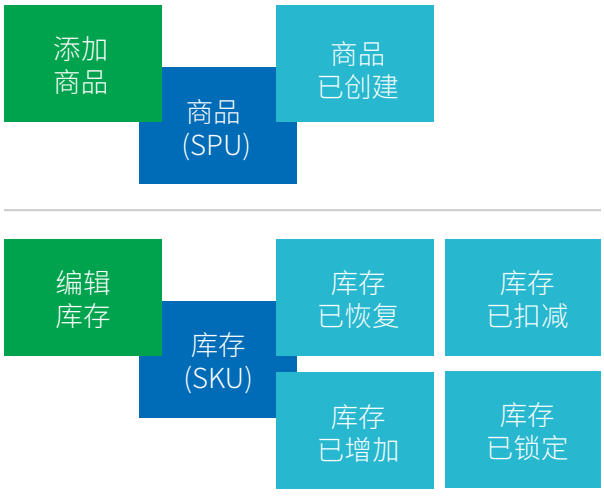
值对象：实体的附加业务概念，用来描述实体所包含的业务信息，无唯一标识，可枚举且不可变，如收货地址、合同种类等等。

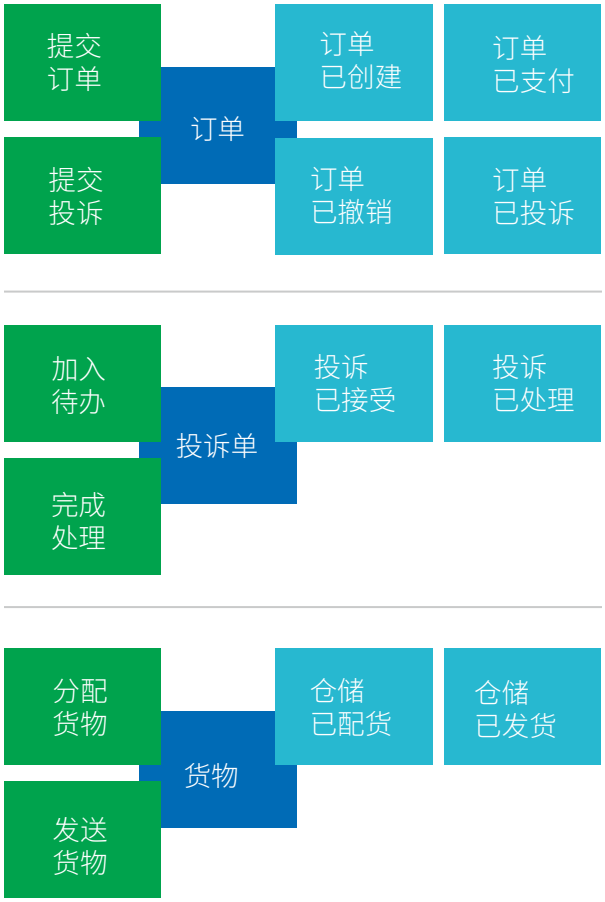
业务架构只负责初步和整体识别领域对象，而对领域对象的分类（聚合根、实体、值对象）和战术层级的详细设计将在应用架构设计部分完成。

领域对象识别的主要步骤如下：

- 对每一个领域事件，快速识别或抽象出与该领域事件最相关（或隐含的）的业务概念，并将其以名词形式予以贴出。
- 检查领域名词和领域事件在概念和粒度（例如数量，单数还是集合）上的一致性，通过重命名的方式统一语言，消除二义性。
- 如果在讨论过程中，有任何因为问题澄清和知识增长带来的对于之前各种产出物的共识性调整，请不要犹豫，立刻予以调整和优化。
- 重复以上步骤，迭代式的完成全部领域对象的识别。

领域对象识别的示例如下：



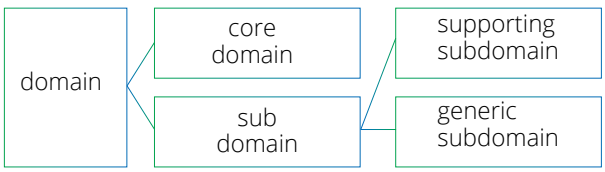


(图 3.2-9 领域对象识别的示例)

3.2.3.2.3 子域划分

子域（Subdomain）：是对问题域的澄清和划分，同时也是对于资源投入优先级的重要参考。比如：“订单子域”、“物流子域”等，子域的划分仍属于业务架构关注范畴。

子域的类型分为：



(图 3.2-10 子域的类型)

核心域（Core Domain）：是当前产品的核心差异化竞争力，是整个业务的盈利来源和基石，如果核心域不存在那么整个业务就不能运作。对于核心域，需要投入最优势的资源（包括能力高的人），和做严谨良好的设计。

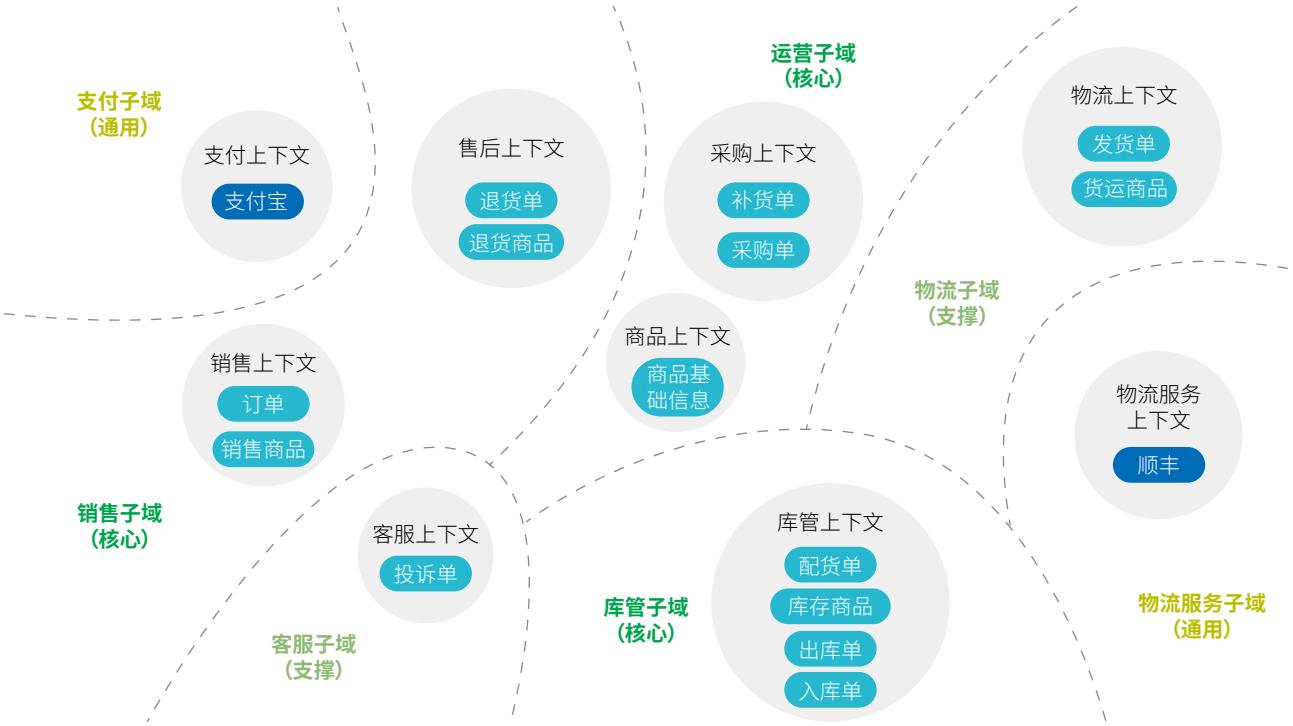
支撑域（Supporting Subdomain）：解决的是支撑核心域运作的问题，其重要程度不如核心域，但具备强烈的个性化需求，难以在业内找到现成的解决方案，需要专门的团队定制开发。

通用域（Generic Subdomain）：该类问题在业内非常常见，所以很可能有现成的解决方案，通过购买或简单修改的方式就可以使用。

子域划分的主要步骤如下：

- 根据“每一个问题子域负责解决一个有独立业务价值的业务问题”的视角出发，可以通过疑问句的方式来澄清和分析子域需要解决的业务问题，例如“如何进行库存管理？（英文描述类似 How to…？）”。
- 利用虚线，将解决同一个业务问题的限界上下文以切割图像的方式划在一起，并以“XXX 子域”的形式对每个子域进行命名。
- 根据三种类型的子域定义，共同结合业务实际（或者参考设计思维中的电梯演讲），确定每个子域的子域类型。

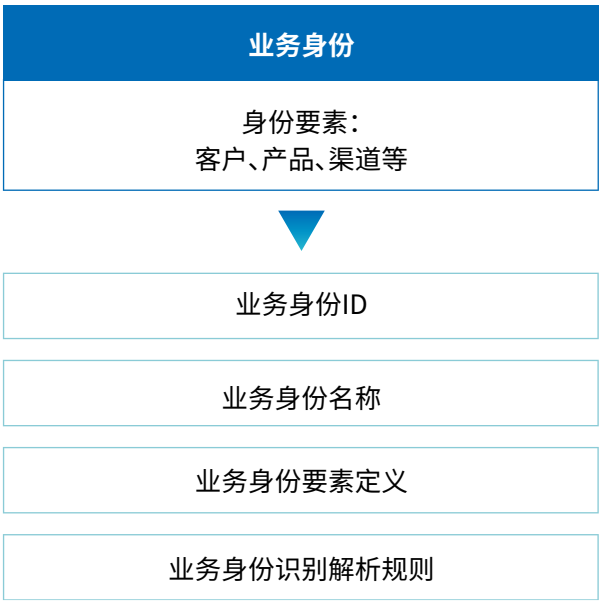
子域划分的示例如下：



（图 3.2-11 子域划分的示例）

3.2.3.3 业务身份建模

业务身份由“业务身份 ID”、“业务身份名称”、“业务身份要素定义”、“业务身份识别解析规则”四个部分组成。



（图 3.2-12 业务身份的组成部分）

其中，业务身份要素定义是最基础、也是最难的部分。企业应根据自身业务特征对身份要素进行识别定义，常见的身份要素维度包括但不限于：客户、产品和渠道等。

业务身份要素除了对要素维度进行抽取识别，还需要定义每个要素维度所包含的领域对象（包括领域对象的属性）；这些领域对象及其属性用来定义业务身份的识别解析规则。实现机制如下：

业务身份 ID			
业务身份名称			
领域对象名称		领域对象 A	
属性 1	属性 2	属性 3	属性 4
“条件规则例如 < 1000”	“条件规则 1（例如 = ‘金卡会员’）”	“条件规则 1（例如 < 100 天）”	…
		条件规则 2	…
	“条件规则 2（例如 = ‘银卡会员’）”	…	…
		…	…
领域对象名称		领域对象 B	
属性 1	属性 2	属性 3	属性 4
条件规则 1	…	…	…
条件规则 2	…	…	…
…	…	…	…

（图 3.2-13 业务身份解析机制）

业务身份建模的主要步骤如下：

- 分析业务组合，提取业务身份要素维度。
- 确定各业务身份要素维度对应的领域对象（包括领域对象的属性）。
- 确定领域对象各属性的取值条件规则，定义业务身份识别解析规则。
- 指定业务身份名称。
- 指定业务身份 ID。

3.2.3.4 能力建模

能力建模分为“基础能力和扩展点设计”、“能力组件设计”和“解决方案设计”三个部分，过程顺序如下：

(图 3.2-14 能力建模的过程顺序)

3.2.3.4.1 基础能力与扩展点设计

基础能力：是对领域对象的原子操作，完成一个领域上单一且完整的职责。比如：创建售后单、修改商品库存量等。

扩展点与扩展实现：“扩展点”是对基础能力的可变性设计，在技术侧体现为基础能力实现中的某一个步骤的接口定义，而接口的一个实现为一个“扩展实现”。比如：订单渲染基础能力中有一个步骤是订单总价试算，而正常时期的总价试算规则与秒杀时期的总价试算规则是不同的，因此需要对订单渲染基础能力设计“订单总价试算规则”的扩展点，并分别定义在正常时期和秒杀时期的扩展实现。

不同的业务通过使用同一个基础能力来达成共享复用的目的，而不同业务在业务规则上的差异性，则通过定义该基础能力在扩展点上的不同扩展实现来区分。

需要注意的是，通常这套机制需要技术上的开发框架支持。

基础能力与扩展点设计的主要步骤如下：

- 对流程建模步骤中输出的各“任务”下的所有“步骤”，确定其对应的领域对象（注意，该领域对象应来自于前面的领域建模步骤）。
- 根据步骤对领域对象的操作，设计对应的基础能力；基础能力的设计应遵循如下原则：
- 完成对一个领域对象单一且完整的操作职责
- 基础能力操作的领域对象最大不能超出单个聚合，最小不能破坏业务的一致性要求
- 基础能力的输入与输出建议对象化，以规范使用
- 根据流程建模步骤中识别出的与该基础能力有关的可变点，以及各业务身份在该可变点上不同的业务规则，提炼设计出基础能力的扩展点
- 确定扩展点的默认实现（即默认情况下基础能力执行的业务规则）

3.2.3.4.2 能力组件设计

能力组件：是对基础能力的进一步封装，目的是方便业务的使用。能力组件加快了业务接入平台的速度，让业务侧专注业务本身，不再需要耗费精力在理解平台大量的基础能力上。

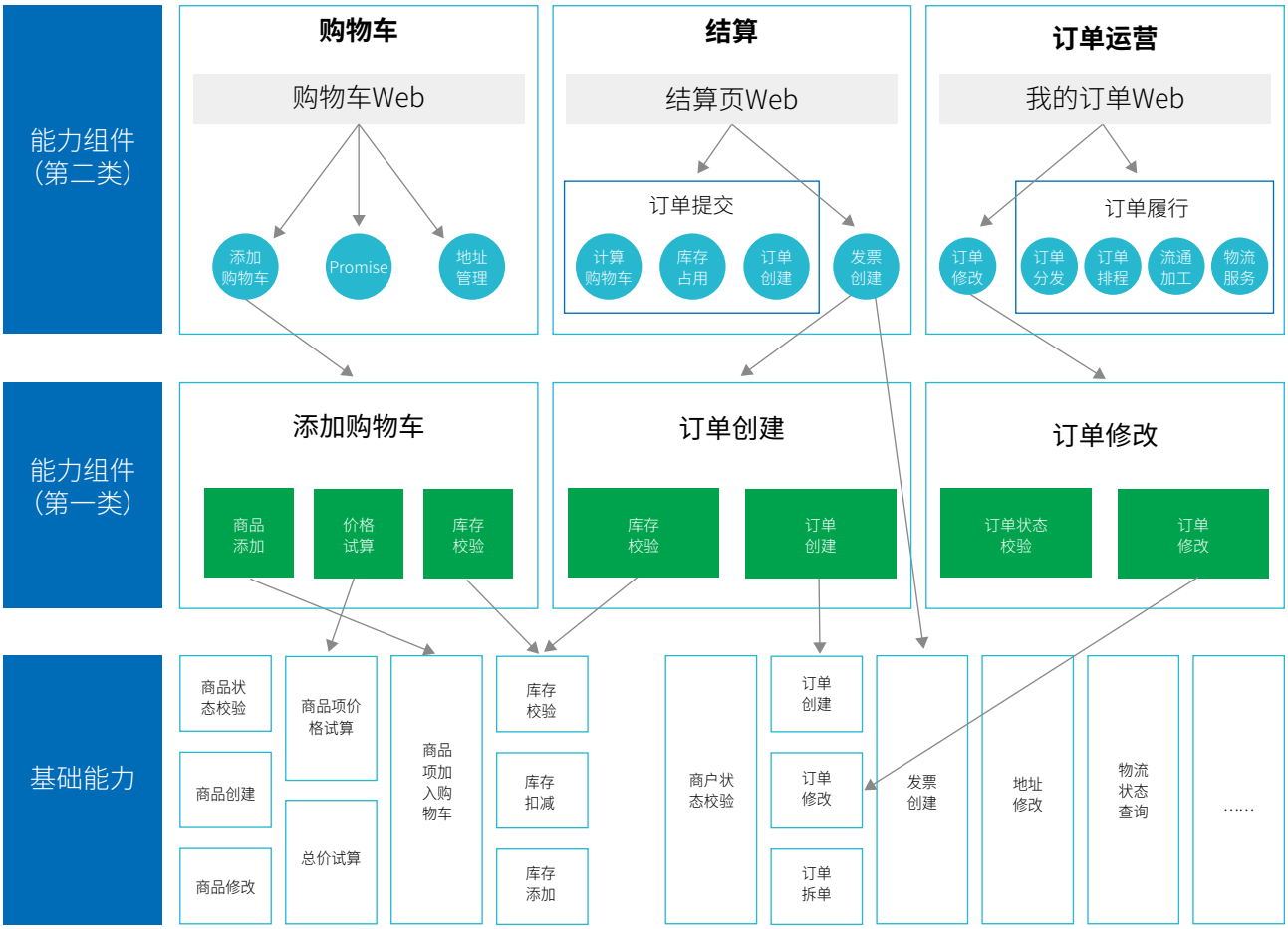
能力组件按封装粒度不同分为两类：

第一类能力组件是根据业务服务的需要编排封装的一组关联的基础能力，从而提供完整的业务服务。

比如：订单创建能力组件。

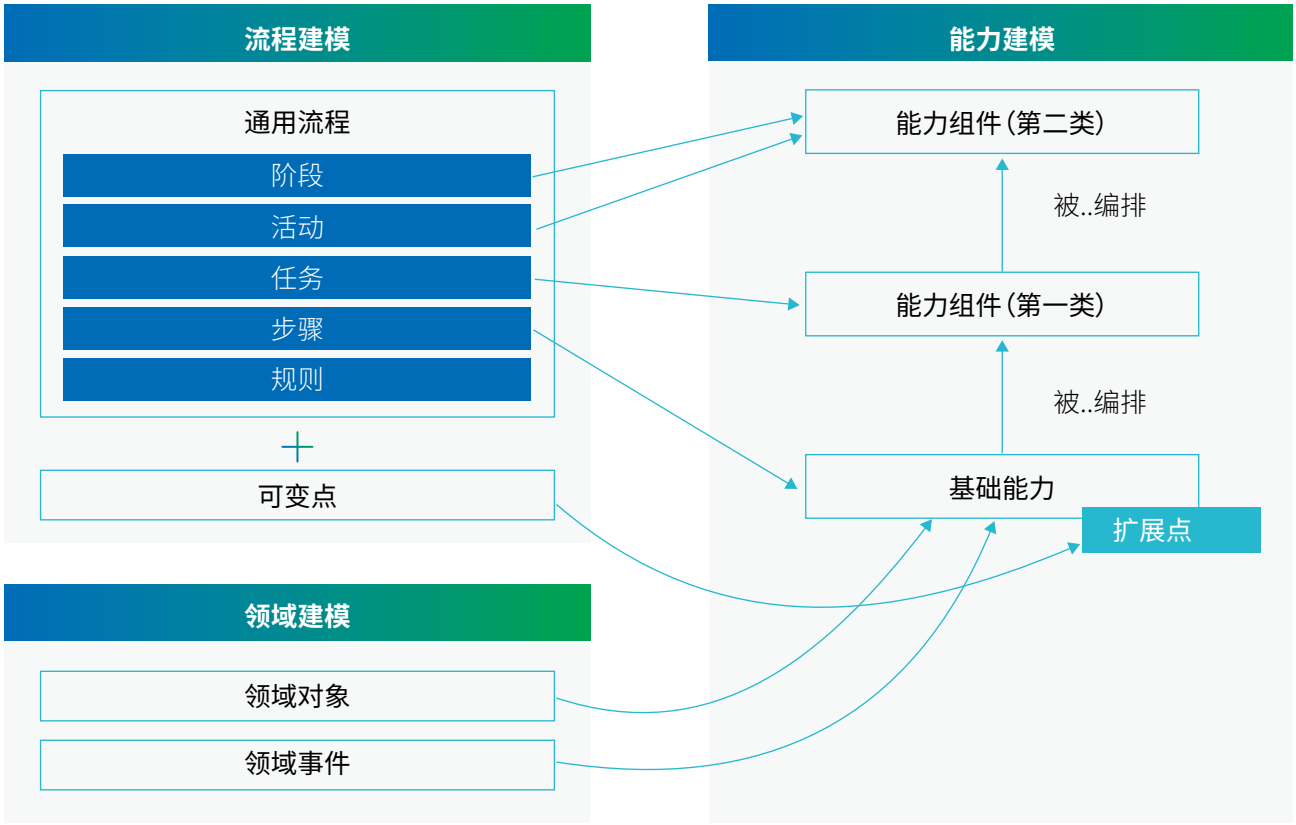
第二类能力组件是平台针对一系列紧密关联的业务活动，设计的能力模板，可基于该模板快速定制某个具体业务的特定流程和能力，从而达到复用全部关联能力的目的。比如：“购物车”、“结算”、“快速建站”等能力组件。

实现机制及示例如下：



（图 3.2-15 能力组件的实现机制及示例）

基础能力与能力组件的设计都基于流程建模和领域建模的结果，各设计产出要素的对应关系如下：



（图 3.2-16 基础能力、能力组件设计与流程和领域建模的关系）

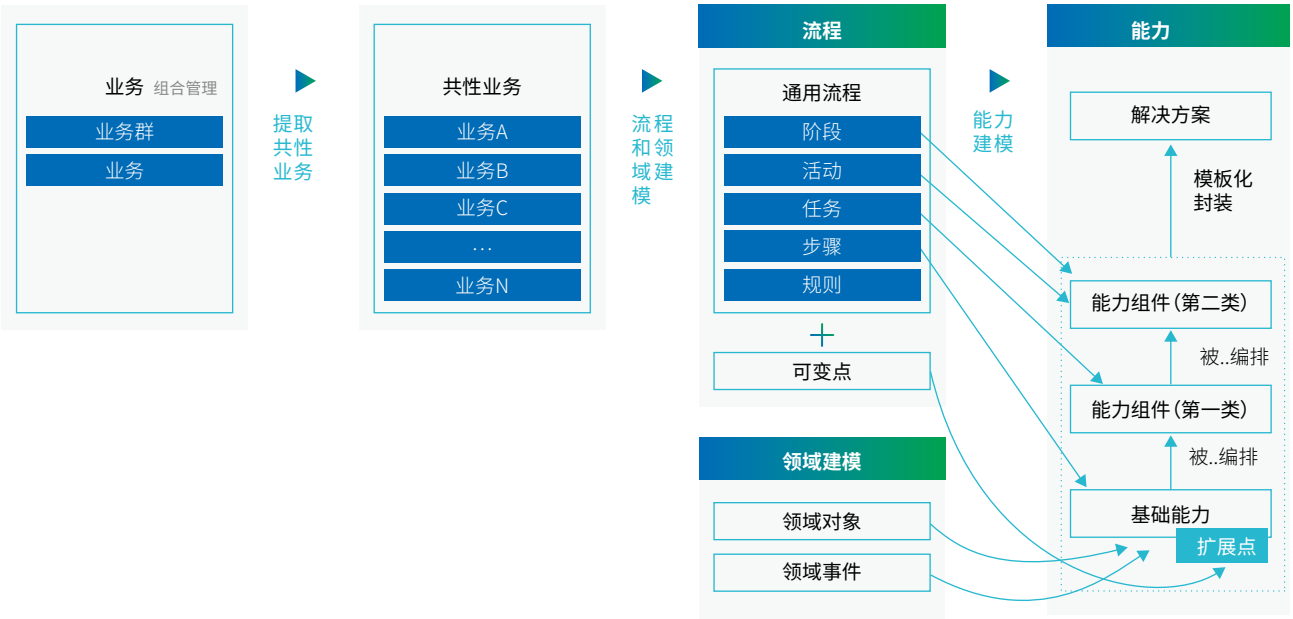
能力组件设计的主要步骤如下：

- 对流程建模中输出的每个任务，设计其所需的业务服务，可采用服务蓝图的方法进行业务服务设计。
- 对每个业务服务，封装编排满足其需求的一组基础能力成为第一类能力组件。
- 对流程建模中输出的阶段和业务活动进行逐项分析，从价值交付和阶段性价值交付的角度，识别对应的一系列紧密关联的业务活动；将这些业务活动包含涉及的所有能力组件和基础能力封装定义为第二类能力组件。

3.2.3.4.3 解决方案设计

解决方案：是平台针对一类共性业务的端到端过程设计的能力模板；可基于该模板快速定制某个具体业务的特定能力，从而达到业务模式复用的目的。比如：虚拟物品交易解决方案。

从以上定义可以看出，解决方案的核心是对共性业务进行识别提取和对业务全部能力进行模板化封装。



（图3.2-17 解决方案设计的过程）

解决方案设计的主要步骤如下：

- 识别和提取共性业务。
- 对共性业务进行流程建模和领域建模，具体方法参见前面所述。
- 进行基础能力和扩展点设计。
- 进行能力组件设计。
- 基于通用流程，将共性业务中包含的所有基础能力、扩展点和能力组件封装定义为解决方案。

3.2.4 如何复用能力，实现新业务快速上线？

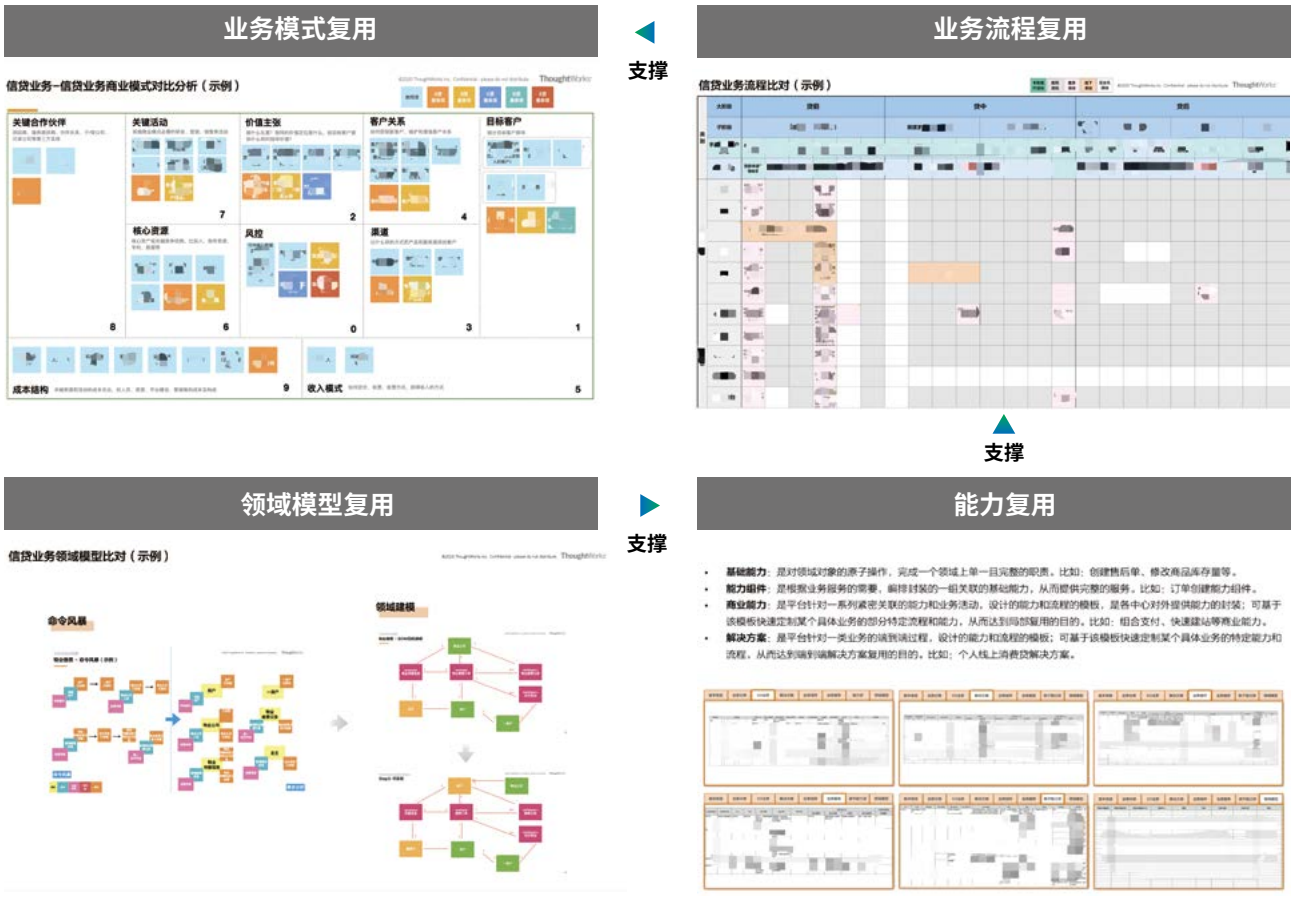
3.2.4.1 多层级复用

在平台化架构中，能力的复用可分为三个层级：

复用层级	复用对象	特征	业务价值
能力共享	基础能力 领域模型	支持同一个业务场景的多个能力消费者，主要解决数据一致性问题。如常见的全渠道策略下，多个渠道客户端使用同一个服务端能力	低
能力复用	业务流程 能力组件 基础能力 领域模型	提供单点的能力，支持不同业务场景的多个能力消费者。由于只提供业务场景中的某一小段，对业务流程的抽象要求适中，需要一定的扩展机制	中
业务模式复用	解决方案 业务流程 能力组件 基础能力 领域模型	提供多个连续或关联的能力，支持不同业务场景的多个能力消费者。对业务流程的抽象要求高，由于需要支持不同业务模式对同一个业务流程的差异化需求，必须有灵活的扩展机制	高

其中“业务能力复用”和“业务模式复用”层级都对可复用能力的识别和设计提出了要求。因此，基于前面论述的可复用能力构建方法，我们就能实现这两层的复用效果。

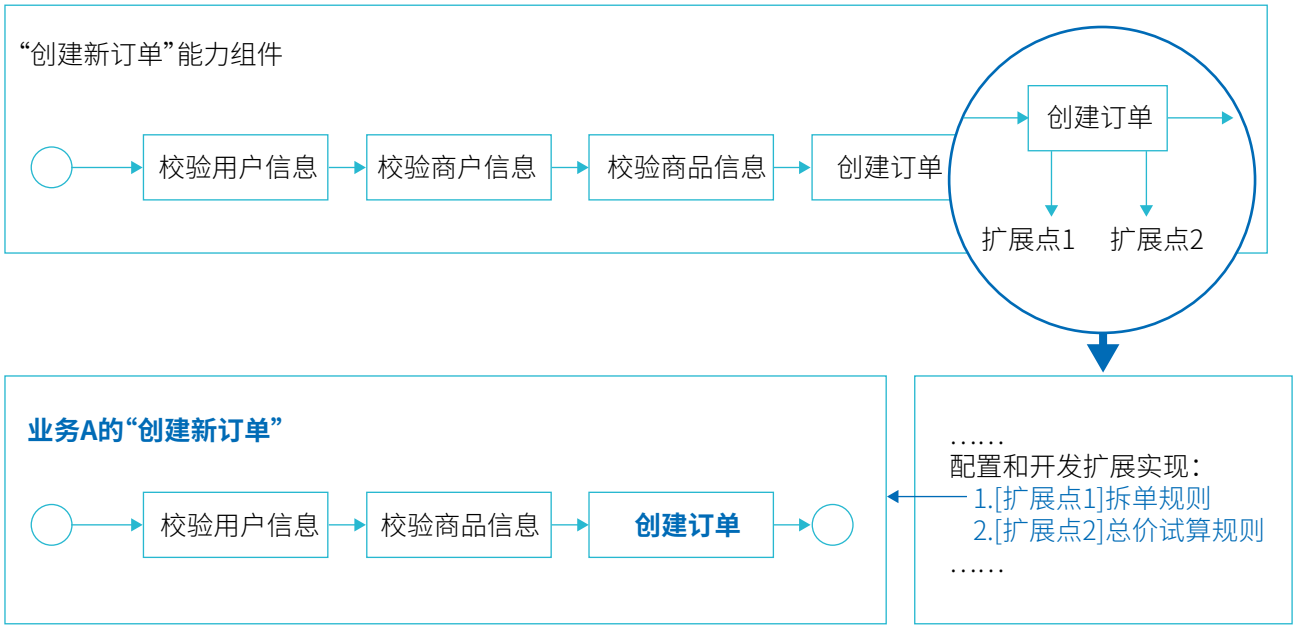
多层级复用为什么能实现呢？首先在于底层领域模型的设计，有了建模后的领域对象提供共享的基础能力，再加上扩展机制，就能实现基础能力在不同业务上的复用；而经过通用流程的建模，这些基础能力又能进一步组装成更大粒度的可复用能力组件，从而实现局部业务流程的复用；如果业务流程的复用能够延伸到业务的全流程，即对于同一类共性业务其全流程都能基于一个解决方案模板定制，而这个解决方案模板是其下所有能力组件和基础能力的封装，那么新的业务只需定制该解决方案模板即可实现业务模式的复用，快速上线新业务。其关系如下：



（图 3.2-18 多层级复用的关系）

3.2.4.2 复用基础能力和能力组件

在识别和构建了平台的基础能力和能力组件之后，不同业务就能针对特定的业务需求复用它们并快速定制。方法是对基础能力下的扩展点进行扩展实现的配置和开发，示例如下：



（图 3.2-19 复用基础能力和能力组件的示例）

复用基础能力和能力组件的主要步骤如下：

- 配置新业务对应的业务身份在各基础能力扩展点上的扩展实现，如果无需定制可直接采用默认扩展实现。
- 开发基础能力的扩展实现。
- 根据业务流程，编排基础能力和能力组件，实现业务需求。
- 对于现有能力不能满足业务需求的情况，需要平台侧新增开发任务，修改或者新增基础能力和能力组件；然后应用侧按上述过程完成定制使用。

3.2.4.3 复用解决方案

对于新业务，如果已构建了其所属共性业务的解决方案，则可通过复用该解决方案进行业务的快速定制。方法是：基于解决方案的通用流程定制新业务流程，并对基础能力下的扩展点进行扩展实现的配置和开发。

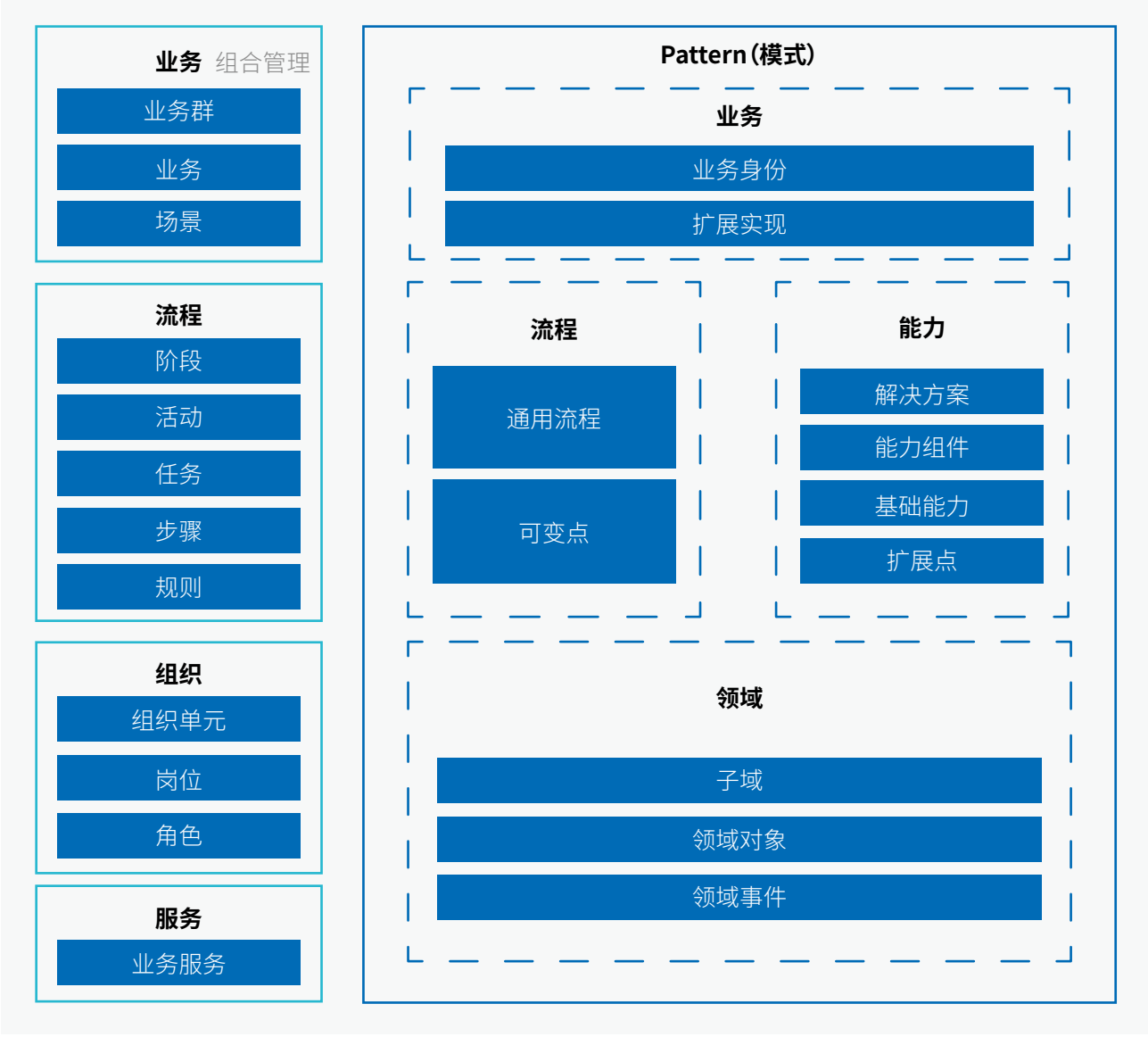
（图 3.2-20 复用解决方案快速定制新业务）

复用解决方案的主要步骤如下：

- 基于选定的解决方案，配置新业务的业务全流程。
- 配置新业务对应的业务身份在各基础能力扩展点上的扩展实现，如果无需定制可直接采用默认扩展实现。
- 开发基础能力的扩展实现。
- 根据业务全流程，编排基础能力和能力组件，实现业务需求。
- 对于现有能力不能满足业务需求的情况，需要平台侧新增开发任务，修改或者新增基础能力、能力组件和解决方案；然后应用侧按上述过程完成定制使用。

3.3 业务架构元模型补充说明

在 3.2 章节中，我们对业务架构元模型的“Pattern（模式）”部分进行了详细说明，下面对业务架构元模型的其余部分进行补充说明；这些部分都可参照传统业务架构的方法进行设计，此处不再赘述。



（图 3.1-1 业务架构元模型）

3.3.1 业务维度

此维度主要对企业的业务组合管理进行建模，分析企业各主营业务和辅助业务的关系结构及运作模式。要素如下：

业务群：是企业基于业务战略拆解，确定开展的特定经营活动。比如：开展 ToC 的电商平台经营活动，其下又进一步细分为快消品业务群和消费电子业务群等。

业务：是业务群下具有业务价值的业务单元。比如：实体商品业务、生活服务业务等；内部支撑类的业务比如人力资源管理等。

场景：是面向用户提供价值的端到端业务场景。通常从 4W1H（What、Who、Where、When、How）的维度识别和定义业务场景要素，然后从业务场景要素的排列组合中筛选出有实际意义的业务场景。

3.3.2 流程维度

此维度主要对企业的业务流程进行分层建模，分析企业如何通过一系列业务活动，按照相关的业务规则将输入转换成为有价值的输出，从而实现用户价值。要素如下：

阶段：即业务流程阶段，包含一组用户的及与用户交互的业务活动，用以实现阶段性价值交付的目的。比如：售前、售中和售后等。

活动：即业务活动，是某个业务角色办理的业务事项，包含一个或一组任务，有明确的业务成果和业务输出。比如：商品发布、活动发布等。

任务：是完成活动的工作程序，是流程的基本组成单元。比如：查询商品详情、更新商品库存、创建订单等。

步骤：是完成任务的具体步骤，是流程的最原子操作。比如：校验用户状态、校验商户状态、订单总价试算等。

业务规则：是定义或约束业务某些方面的陈述，旨在维护业务结构或控制或影响业务行为，它描述业务过程与决策过程。比如：if 订单 . 提货方式 = “邮寄” 且 订单 . 支付状态 = “已支付” then “创建物流单”。

3.3.3 组织维度

此维度主要对企业的组织体系进行分析。公司组织体系指为了保障战略和业务规划落地，以及有效实施业务流程体系，公司采取的组织架构和管控模式。要素如下：

组织单元：是公司组织体系的组成单元。

岗位：是随组织结构设定的，要求个体完成的一项或多项责任以及为此赋予个体的权力的总和。

角色：是业务流程中活动参与者的原型，参与者在流程中的位置通过担任合适的角色确定。组织为完成某一目标，往往会把此目标分解，以便能交给不同能力和责任的角色合作完成。

3.3.4 服务维度

此维度主要对企业对内和对外提供的业务服务进行建模分析。要素如下：

业务服务：是企业和企业的每个业务单元为其客户提供的内部和外部服务。服务是能力对外呈现价值的方式，是具备明确的业务特征，独立完整、由一个或多个关联紧密的功能组合的集合。比如：开户、提交订单等。

4

现代企业架构框架 — 应用架构

应用架构的核心关注点是业务需求是由哪些应用承载的，它们与用户是如何交互的，它们之间的关系以及是如何交互的，它们访问或变更了什么数据。

应用架构的设计主要以应用（Application）的设计为核心，向外围可以延伸到平台型企业架构对于应用分层，分组的设计。例如大家关注的以微服务为代表的分布式应用架构，以及此类架构模式下的常见问题，例如微服务如何划分如何组织，都是应用架构在这个粒度需要关注的问题。

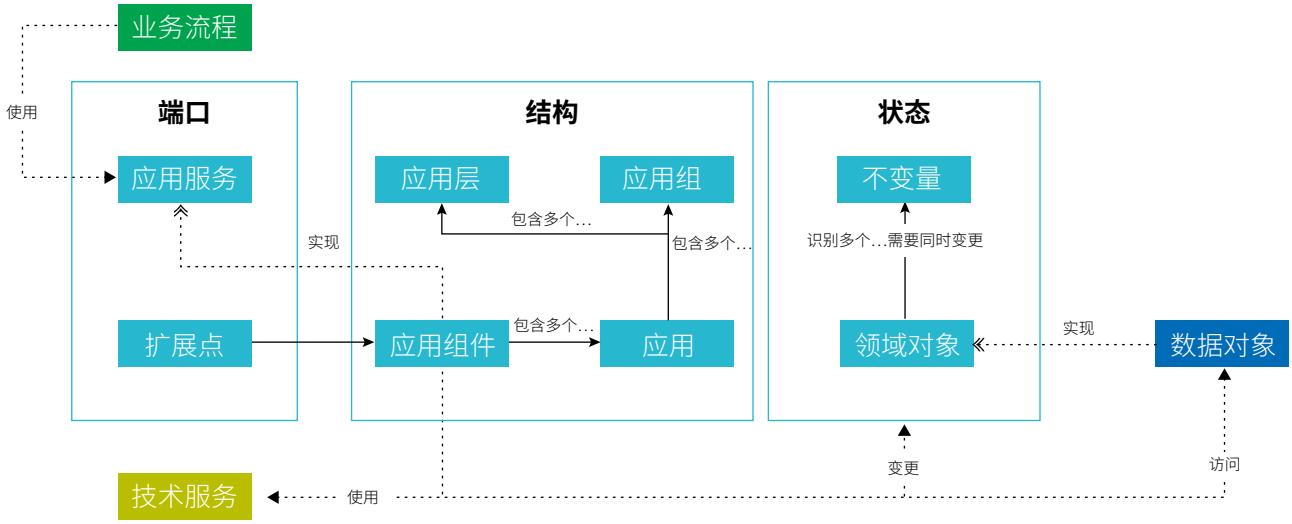
同样，以应用为基准，向内部延伸又会涉及到应用内部的架构设计。例如常见的应用分层设计，领域驱动设计中提到的六边形架构、洋葱模型，包括领域对象的详细建模与设计，都是在应用架构这个粒度需要关注的问题。

而其中的领域对象设计在业务架构以及后续的数据架构中都会提及，本框架充分融合了企业架构与领域驱动设计的思想和方法，从业务架构到应用架构以及后续展开的数据架构，都秉承以领域对象设计作为架构的核心要素，跨越架构边界，使领域对象作为一条主线，串联起各个架构视图，也有利于保证各类架构的连贯和一致性。

4.1 应用架构元模型综述

应用架构的内容元模型包括“端口”、“结构”、“状态”三个部分，如下图所示：

- 结构部分用来对 IT 系统的职责、边界建模，其中包括，应用组件、应用、应用组、应用层
- 端口部分用来对应用的出入口建模，其中包括应用服务和扩展点
- 状态部分用来对应用状态的变更建模，其中包括领域对象和不变量



(图 4.1-1 企业级应用架构元模型)

4.2 应用架构元模型应用

4.2.1 平台化趋势对应用架构提出的新挑战

平台化趋势意味着企业 IT 系统的形态逐渐从扁平结构转向分层结构，即一部分应用提供可复用的能力，组成底部的平台，而其他应用建立在平台之上。除了能力复用外，另一个提升 IT 支撑响应力的关键是，提升 IT 系统各组成部分的自治性，使得变更能够相对独立的、以小步快跑的方式发生。因为无论是创新也好，集中管控也好，虽然变化速率不同，但变化始终存在，既然变化不可避免，我们应将精力投入到应对变化上。

在我们的经验中，应用边界划分不合理会对应对变化产生明显的负面作用：

	粒度过大	粒度过细
功能维度	• 多个团队在同一个物理边界内，变更计划和所需资源容易冲突，协作复杂 • 变更实现难度高，质量保障困难或耗时	• 若一个变更常常散布在多个物理边界内，集成、测试、部署负担重、耗时，且常常伴随着高成本的跨团队沟通协调
运行维度	• 受限于物理边界，无法将故障隔离在局部区域，因故障级联影响业务支撑 • 受限于物理边界，无法按需调整应用的容量，造成资源浪费	• 基础设施升级、变更工作量大，或对团队自动化运维能力要求高 • 平添大量网络通信的开销，反而降低整体性能

这些负面作用会拖慢 IT 支撑的响应力或稳定性，因此，“如何划分 IT 系统的边界，以合理的布局更好地应对变化”是需要解决的挑战。在平台化趋势之前就已经开始流行的微服务架构风格的催化下，这个挑战就已凸显，而平台化强调可复用的服务，必然会对原有系统进行打散和重组，进一步加剧了这个挑战。

4.2.2 如何划分 IT 系统的边界，以合理的布局更好地应对变化

从上文的分析可以看出，边界划分其实从应用架构视角出发，对功能、运行层面变化的应对设计，是应用架构设计的重要部分。我们在进行应用架构设计的过程中，融合了领域驱动设计中的限界上下文与统一语言的概念，延续业务架构部分中对于领域对象的识别和子域划分，结合组织与技术的要素，从多个方面充分考量对于应用的建模。

限界上下文（Boundary Context）：是业务上下文的边界，在该边界内，当我们去交流某个业务概念时，不会产生理解和认知上的歧义（二义性），限界上下文是统一语言的重要保证。

统一语言（Ubiquitous Language）：是 Eric Evans 在《域驱动设计 - 处理软件核心中的复杂性》中使用的术语，用于构建由团队，开发人员，领域专家和其他参与者共享的语言，也称为无处不在的语言、通用语言、泛在语言。统一语言是在限界上下文中建模的，在其中标识表达了业务领域的术语和概念，并且不应该有歧义。

带着对于业务上下文边界的理解，使用业务、技术都能理解的统一语言，以两大阶段实现边界划分：

1. 从功能需求层面出发，按“单一职责”原则划分逻辑边界
2. 加入非功能需求，按架构质量属性调整逻辑边界并划分物理边界

这个设计过程可通过元模型提供的应用层、应用组、应用、应用组件进行不同层次的边界划分建模。

4.2.2.1 应用组建模

应用组是一种大比例结构的逻辑边界划分结构，其主要作用是：

- 应用组作为一种粗粒度的划分，帮助我们快速找到进一步划分的切入点
- 在微服务化的大背景下，物理边界逐步碎片化，我们在与利益干系人对话时，需要一个能够代表一组相关物理边界的结构，以避免不必要的细节干扰

在结构上，应用组包含了多个应用（物理边界）。应用组常常对应到数字化产品级别，例如 CRM 系统（客户关系管理系统，其下可能包括客户档案管理应用、客户忠诚度管理应用）；在业界流行的按中心划分的中台 / 平台架构里，应用组常常对应到某个中心。

应用组的建模依据主要来自于业务架构的业务、组织两部分成果的输入。在从 0 到 1 的应用架构设计场景里，这个步骤不求精确，主要是帮助我们建立划分的起点。例如：

> 汽车行业的经销商，会同时开展新车销售和售后服务两个不同的业务，在经销商内部一般也由不同的组织负责。

> 而维修保养和配件销售又是售后服务中的两个不同业务场景。
>
> 我们可依此快速建立一个两级的应用组，这个结构并不精确，但足够我们进行下一步工作。

4.2.2.2 应用组件建模

应用组件是一种细粒度的应用逻辑边界划分结构，是对功能、数据的封装。

向上，一个或多个应用组件可组成一个应用（物理边界），向下，就是代码实现层面的结构设计了。因此，应用组件是架构层面运用“单一职责”原则的最小单元。

按职责类型分解，应用组件可分为四种常见的参考类型：

类型	功能	数据	例子	变化原因
流转类	支撑某个流程或流程中的一段活动	流程的状态流转信息和决策证据	结账组件，其支撑的流程是结账流程，管理的数据是订单	流程编排变化
规格类	支撑某个业务规则，给出专业意见	一般不管理数据的生命周期，只负责加工给出结果	• 验证：如发票是否逾期 • 筛选：三月内消费过的客户 • 计算：计价、导航路线建议	业务规则的计算逻辑变化
视图类	支撑某个业务活动所需的整合信息	一般不管理数据的生命周期，只负责加工给出结果	• 商品信息的整合展示 • 统计报表	视图组装逻辑变化
配置类	为前三类提供配置输入	基础的资源信息	商品目录设置内容管理系统中的配置功能规则的开关	前三类对配置要求的变化

一般来说，我们建议从流转类的应用组件开始识别，再延伸至规格类、视图类，最后再识别配置类。建模依据主要来自于业务架构的业务流程、业务对象、业务规则。其过程可总结为三个子步骤：

子步骤一：功能和数据识别

逐层分解业务流程，从活动、任务、步骤中可以得到相对细粒度的业务需求，即 IT 系统需要提供的功能；将业务对象转化为不同类型的数据对象。

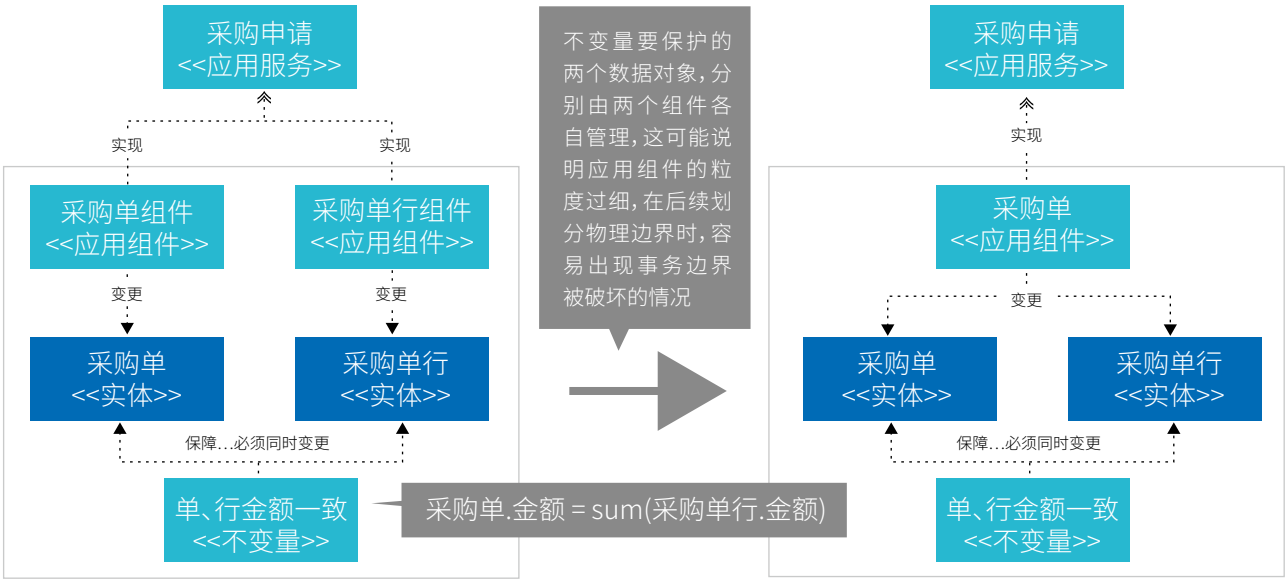
子步骤二：功能与数据关联，识别应用组件

按不同组件类型将功能与数据对象关联对应，得出应用组件。在流转类组件建模时，有两个关注点需要特别注意：

一是隐式业务边界的识别。在对业务流程分析时，地点、角色变化时，我们面对的业务干系人和他们的工作环境不同，其关注点可能不同，这往往会成为不同的变化原因。通过识别这类隐式的业务边界，这可以帮助我们细分数据对象和应用组件，例如：

> 线上订餐流程中，客户提交订单并完成付款后，订单会被派单至厨房备餐。即使在业务架构的产出中没有识别出客户订单和厨房备餐单据可能是不同的业务对象，也可以从地点和角色的变化将订餐结账、备餐识别为不同的应用组件

二是数据对象变更一致性约束的识别。在流转类应用组件的建模过程中，应该尽可能识别业务规则中对于数据对象变更的一致性约束，以元模型中的“不



(图 4.2-2 服务、组件、数据、不变量的关联)

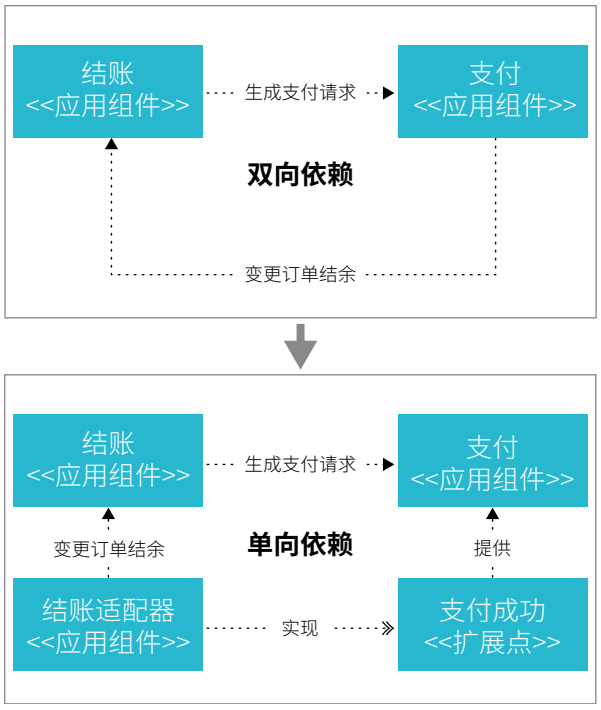
变量”建模。不变量代表了哪些数据对象需要被同时改变，我们可以依此复查，是否将应用组件拆分得太细，破坏了事务边界。

子步骤三：识别、调整各应用组件之间的依赖关系

识别各组件间的依赖关系，在这里尽可能避免两种依赖：

一是尽可能避免依赖容易变化的组件。尽管可以定义契约，但容易变化的组件容易打破先前定义的契约。

二是尽可能避免出现双向依赖。双向依赖往往对可适配性产生负面影响，可通过引入第三个组件或扩展点（详见“应用服务与扩展点”）解耦。



（图 4.2-3 通过扩展点将双向依赖转换为单向依赖）

4.2.2.3 应用建模

有了应用组件作为“原料”，应用架构的物理边界设计就有了输入，下一步的目标是识别是否存在架构质量属性冲突，作为应用建模的重要参考，调整逻辑边界，并确定物理边界。质量属性冲突包括：

- **该边界内是否存在变更频率冲突**

我们倾向于将高频变更的应用组件与其他应用组件阻隔开。物理边界意味着部署的独立性，不容易产生发布计划、部署所需资源上的冲突。如果一个功能扩展带来的变更，集中在某个应用、甚至应用组件内，其协作成本相对更低。

- **该边界内是否有特别的移植性要求**

该要求指应用迁移到不同组织、业务运作方式的能力，或者换一个角度来说是应用承接新业务模式的能力，这恰巧是平台所需要的关键能力。在实现某一个业务模式时，必然存在该业务模式的特定需求和可以支持多个业务模式的公共需求。我们倾向于将特定需求和公共需求的实现隔离到不同的应用组件、甚至应用里。以便新业务模式入住时，方便实现功能扩展以及实现部署要求。

- **该边界内是否有特别的弹性要求**

该要求指应用是否容易通过调整容量来应对流量变化。在这里应用的粒度决定了容量调整的难度和成本。因为应用的粒度太大，而流量变化只影响其中某个应用组件，则扩容带来了不必要的成本浪费。另一种情况是某个应用组件不能调整容量或者非常

困难，这主要是因为其依赖于某个无法扩展的技术组件。例如，某应用组件依赖硬件加密设备（技术组件）对报文进行加密，扩容意味着需要采买硬件加密设备。因此，需要将这类应用组件隔离开，使其他组件更容易应对弹性要求。

- **该边界内是否有特别的性能要求**

该要求与弹性要求类似，将不同性能要求（请求处理的快慢程度）的应用组件分开，特别是对于特定问题，可能适合某个技术栈，但出于整体建设、运维成本考虑并适合大面积使用，我们倾向于将其设计为一个独立的应用，与其他应用组件隔离开，使得异构。典型的例子是，部分高性能场景适合用 C 语言实现。

- **该边界内是否有特别的可用性要求**

该要求与弹性要求类似，将不同可用性要求的应用组件分开，降低保障可用性要求的建设、运维成本。例如，如果某个应用组件支撑的功能尚处在业务探索阶段，那么可以适当降低其可用性要求，这要求将其与承担核心功能的应用组件隔开。或是当故障发生时，能否将故障隔离在局部范围，减少应用失效造成的业务损失。

- **该边界内是否有特别的信息安全要求**

例如，某应用组件需要保管信用卡持卡人信息，为降低该信息被非授权访问的风险，我们倾向于将该应用组件与其他应用组件拆开，将其独立部署在一个加固的运行环境中。

- **该边界内是否有特别的合规要求**

有时边界划分会受到法律、法规或行业规定的影响。例如某业务是需要公证的抽奖活动，按照当地行业要求，中奖人抽取的实现需要通过公证处认证。我们倾向于由已通过认证的外部应用服务来提供该职责，或是将该职责独立划分为一个应用，减少认证需要花费的时间。

基于以上冲突的进行逻辑边界调整，将由特别要求的应用组件独立划分到各自的物理边界，剩余的应用组件原则上保留在一个物理边界，以元模型中的“应用”表述该划分结果。

4.2.2.4 应用层建模

除了应用组之外，常见的一种大比例结构是分层，因此我们也将应用层作为一种元素加入进来。我们认为分层代表了企业对变化速率的认知，并为不同的变化速率匹配架构设计目标和管理方法。并不是所有的 IT 系统都需要“最高配”的架构属性，实现成本也是重要的约束条件。一个处在业务探索期的信息体统，其生命周期一般只有 6-12 个月，且变化剧烈。为其达成过高的架构属性显然是不具备投资合理性的。

另一方面，平台化架构中作为支撑层的 IT 系统在架构属性上需要更多重视和投入。我们常常看到企业仅仅在全景图纸上做了分层，但缺少架构设计、治理中的实际举措，功能上的变更往往会击穿多层，支撑层团队疲于奔命。例如：

> 企业有多条业务线，各自有不同商品结构，原各设置一个前台团队负责其应用的交付和运营。
>
> 为降低成本，决定将各业务线的商品展示、搜索等需求集中起来，设立商品中心作为平台支撑层的应用 / 应用组。
>
> 这个初衷是好的，但如果商品中心仅仅是“复刻”各业务线的商品结构，而未作相应设计的话，面对多条业务线的多线需求，往往不堪重负，成为瓶颈。

因此商品中心必须设计一个商品结构的元模型，可通过运营人员组装的方式，为不同业务线的商品结构建模。这样才能将前台各业务线的商品展示、搜索需求变化就地消化。

因此，如果对分层做出设计，那么在职责划分上就要做出应对，否则分层容易成为空中楼阁。

4.2.2.5 应用服务与扩展点建模

元模型中的应用服务最主要的作用是显式地向对外定义一个契约。这在做应用组件升级 / 替换、定义 IT 服务级别等架构治理场景中会有帮助。应用服务可用来对一组相关的应用组件功能集合建模，例如:

> 在线上订餐流程中，用户需要 IT 系统支持提交订单、发起付款、订单状态查看、取消订单四个行为，可将它们建模为应用服务——订餐结账

应用服务的来源可以是“自动化”的业务服务，如果在业务架构采用了“能力建模”（见 3.2.3.4 能力建模）也可以直接将其构建块转换过来：

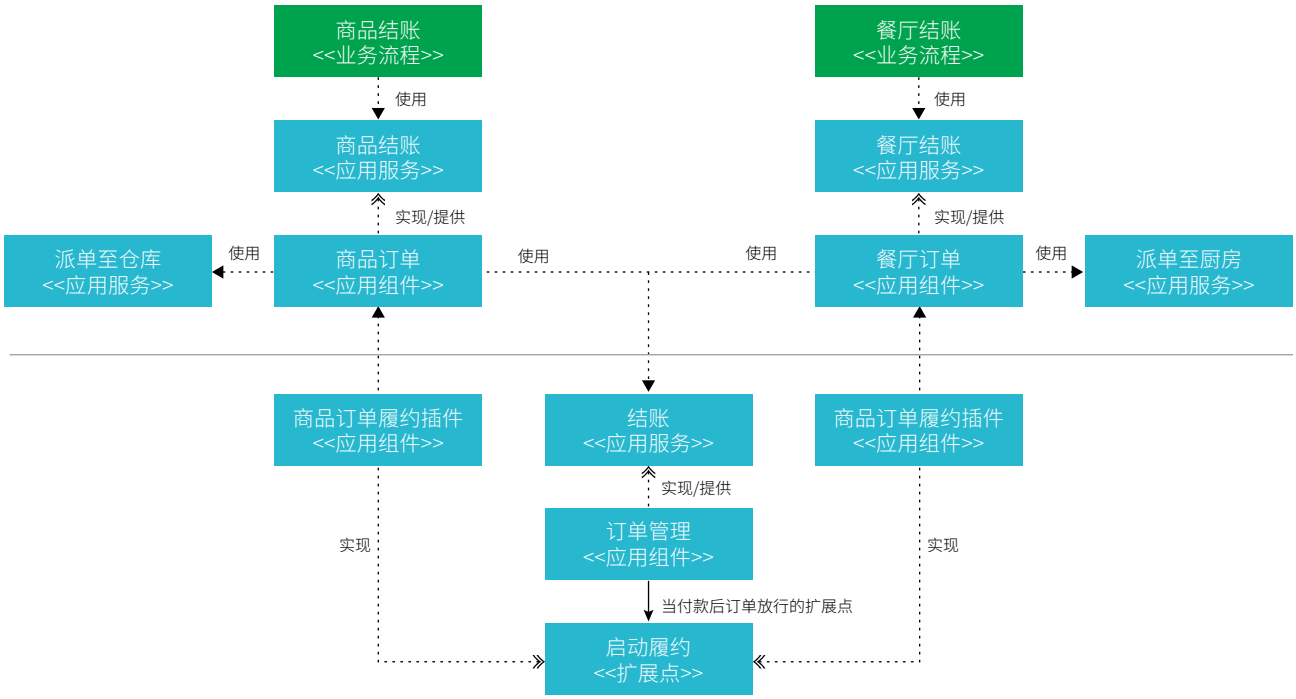
- “能力组件”转换为“应用服务”
- “基础能力”转换为“功能”

另一方面，如果说应用服务显式定义了应用组件的入口，那么扩展点则显式定义了应用组件的出口。扩展点有两个作用：

一是反转对不稳定应用组件的依赖，降低其变化对自身的冲击，这在“应用组件建模”一节中有提到。

二是增强可移植性质量属性，即不同的业务场景下对于同一个应用组件的逻辑存在差异化需求，对业务架构的“变化点”，根据当前应用状态上下文中的业务身份，针对性选择适合的逻辑实现。例如：

> 某公司有零售门店，销售零售商品，同时在店内还有餐厅，我们可以将客户结账识别为一个可共享服务。结账后，商品订单需要派单至仓库提货，而餐厅订单需要派单至厨房备餐，这需要使用“放行履约”这个扩展点，并找到两种业务模式的扩展实现关系。



(图 4.2-4 典型的扩展点建模结果)

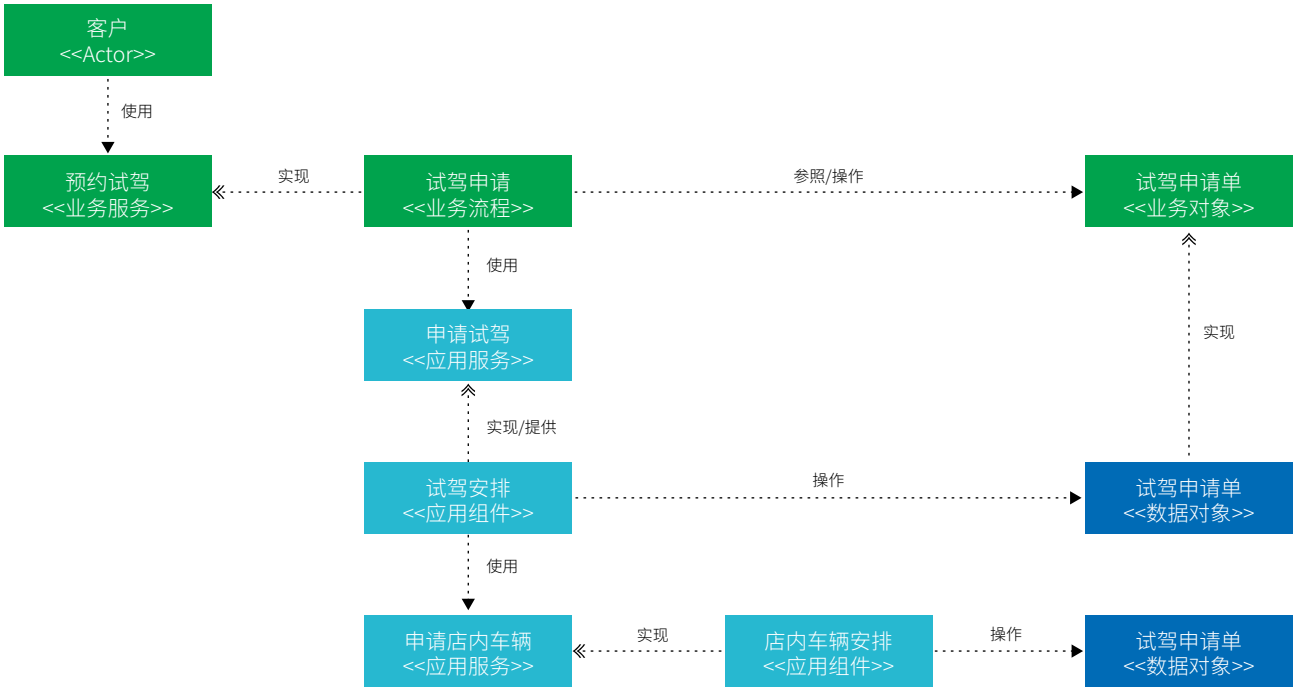
总结来说，应用服务和扩展点是对边界概念的加强，帮助我们理解跨边界的交互。

4.2.2.6 边界划分结果和依据的可视化

我们建议将边界划分的结果及过程依据保存下来并可以开放给授权人员访问。这是因为我们常常见到这样的问题：“新的功能应该放在哪个应用实现？”

这个问题背后的原因可能是应用的边界划分不清晰，职责模糊，或者是边界划分的结果及依据丢失了。常见的现象是我们看到一张张应用架构全景图，由若干个方框组成，代表一个应用或应用组件。由于缺少上下文，仅凭方框内的名字很难判断应用的职责范围，所以不好回答“**新的功能应该放在哪个应用实现**”这样的问题。

解决这个问题的难点是如何简练、清晰地描述应用的边界和职责。在全景图的基础上，为每个应用 / 应用组件增加职责描述是一个不错的起点，但仅用文字描述可能存在歧义。我们建议通过建立应用架构与业务架构、数据架构的构建块映射来解决这个问题。



(图 4.2-5 典型的业务架构 - 应用架构 - 数据架构的构建块映射关系)

通过构建块映射关系（业务流程使用应用服务，应用服务由应用组件提供，应用组件操作数据对象），应用 / 应用组件在业务活动中的职责有了明确的表达，再配合文字来描述引导阅读和理解，效果更佳。我们推荐为每个应用组 / 应用 / 应用组件建立自己的主页，将其与其他元素的映射关系作为内容显示地展示出来。

最后，应用架构阶段更像是在为 IT 系统应该建设成什么样子提出要求，所以应用架构设计应该是和技术实现方案解耦的（虽然技术的升级可能使得应用架构的设计风格产生变化），从而将技术变化隔离在可控范围内。

组件名称 试驾安排

职责定位	相关业务活动	所有者
自动化试驾申请流程，支持线上、线下渠道试驾申请的统一管理	• 试驾申请流程 • 零售线索跟进流程	• 零售产品组-1
实现的应用服务	**依赖的应用服务**	**其它链接**
• 申请试驾	• 规划试驾路线 • 申请店内车辆 • 零售线索跟进	• 项目需求管理 • 项目代码仓库 • 项目部署流水线
管理的数据对象	**提供的扩展点**	
• 试驾申请单	• 试驾项目参与门槛验证	

(图 4.2-6 典型的应用 / 应用组件画布)

5

现代企业架构框架 — 数据架构

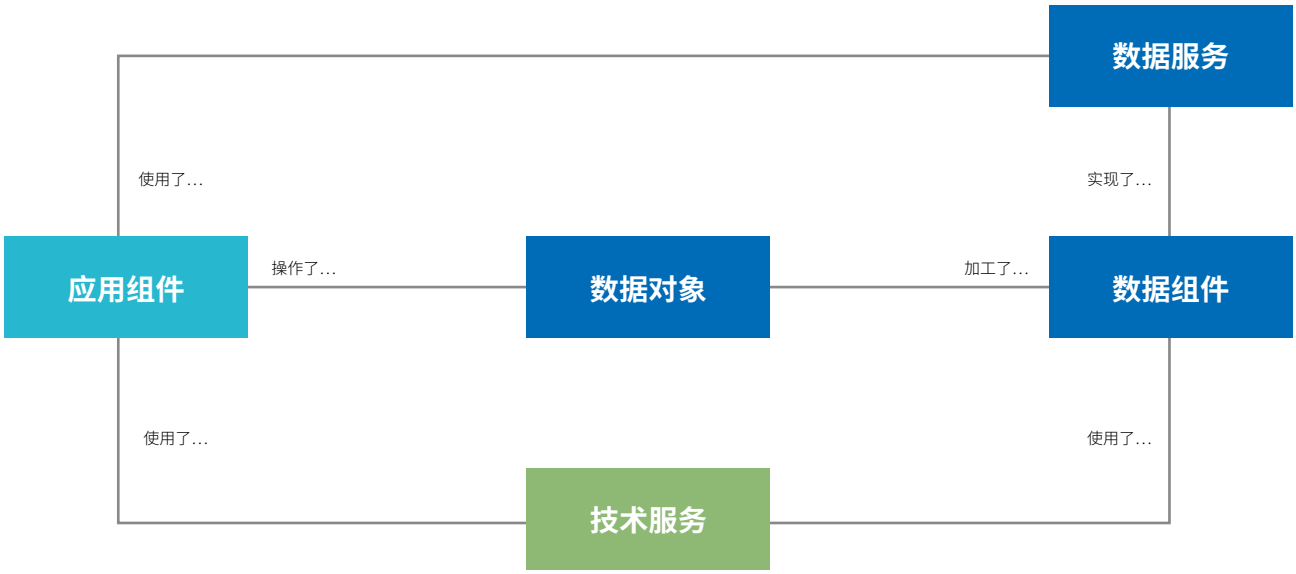
数据架构描述的是企业经营过程中所需数据的结构及其管理方法，其目标是将业务需要转换为数据需求。

值得一提的是，数据架构不等于数据中台，数据中台是一种企业架构设计的整体结果，包含了不同视角（业务、应用，数据、技术），而数据架构是数据视角。良好的数据架构规划和设计，为数据中台以及所代表的数据驱动运营、数据驱动业务都奠定了好的基础。

5.1 数据架构元模型综述

数据架构的内容元模型包括“结构”、“端口”两个部分，如下图所示：

- 结构部分用来对数据模型、数据处理建模，其中包括数据对象、数据组件
- 端口部分用来对数据模型的边界建模，其中包括数据服务



(图 5.1-1 企业级数据架构元模型)

5.2 数据架构元模型应用

5.2.1 平台化趋势对数据架构提出的新挑战

长久以来，业界对数据架构的通用做法是：对于运行类 (Operational) 场景和分析类 (Analytical) 场景，应该使用不同的设计方法和技术支撑。

运行类场景以业务事务为主线，关注点是业务事务运作证据的完整性和一致性，以及确保各类数据在各业务单元间高效、准确地传递，实现跨业务单元的事务推进，其设计方法和技术已经沉淀了很多年。

运行类场景的数据架构设计，目前的关注点在于分布式架构下，如何建立企业级一致的数据标准体系、数据所有权定义以及数据自描述能力，为企业级的数据治理以及对于分析类场景的支持奠定基础。

分析类场景则需要对内、外部数据进行收集和加工，用来评估业务运行表现（度量、分析、预测），或者结合机器学习技术给出对于未来发展趋势预测和判断，尝试构建数据驱动运营的企业组织。其设计方法和技术可分为数据仓库、数据湖两个方向，它们有各自不同的适用场景和技术栈，但共同点是与运行类场景有显著的不同。

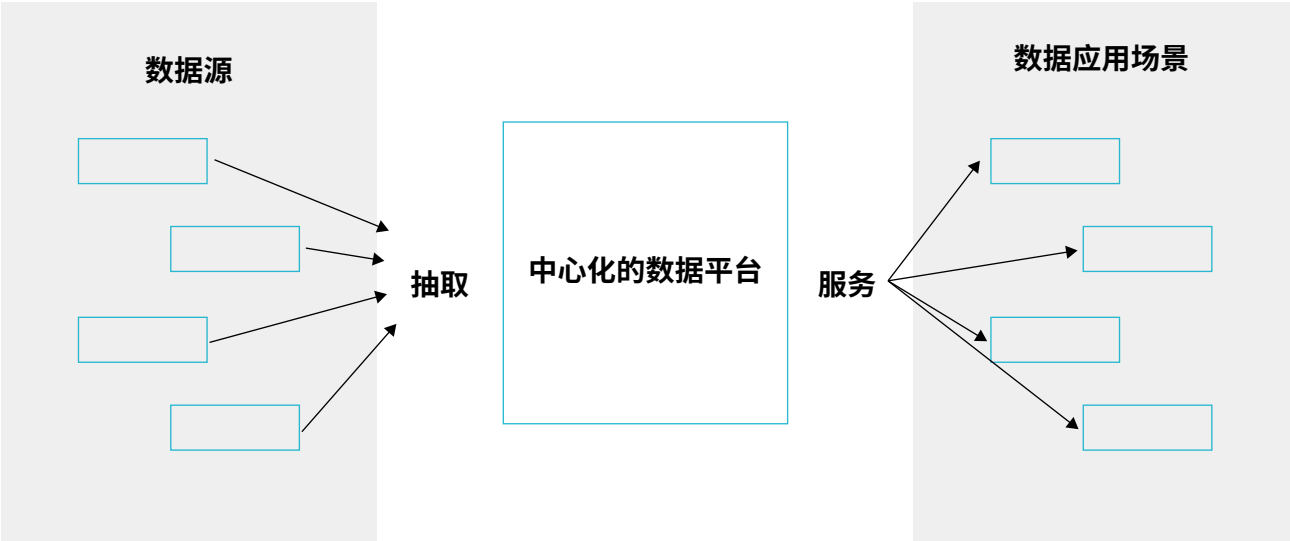
为此，许多企业组建了专职的集中式数据团队，将分析类数据处理工作和其背后的复杂性打包成为一个黑盒，提供端到端的统一的数据类企业级服务与支撑。当我们从高处鸟瞰，往往可以看到为一个中心化的数据架构。

这个模式对于业务场景简单的企业环境工作得不错，但对于多业务线、业务平台化的企业环境已初显疲态。一方面，随着 IT 建设加速，数据源和分析类场景的数量激增，对数据服务的响应力提出了更高要求。另一方面，想要提供高质量的数据服务，除了分析类数据的专业技能，还要求对于业务场景、现有应用软件的深入理解。如果所有工作仍然只由专职的集中式团队一肩挑，团队带宽的限制必然会拖慢响应力。

因此，我们认为需要探索的是如何适当拆分过于集中的分析类数据处理职责，为专职数据团队减负，使其可以将精力投入到高价值的分析类场景中。

5.2.2 如何适当拆分过于集中的分析类数据处理职责，缓解规模化瓶颈

要回答这个问题，需要先打开原先的数据服务黑盒。不管是该服务的实现方案基于数据仓库还是基于数



（图 5.2-2 从全景鸟瞰中心化的数据架构）

据湖，数据想要形成分析类价值，背后需要经过摄取（Ingest）- 加工（Process）- 能力包装（Serve）三大工序，数据架构元模型可以帮助我们对此建模。

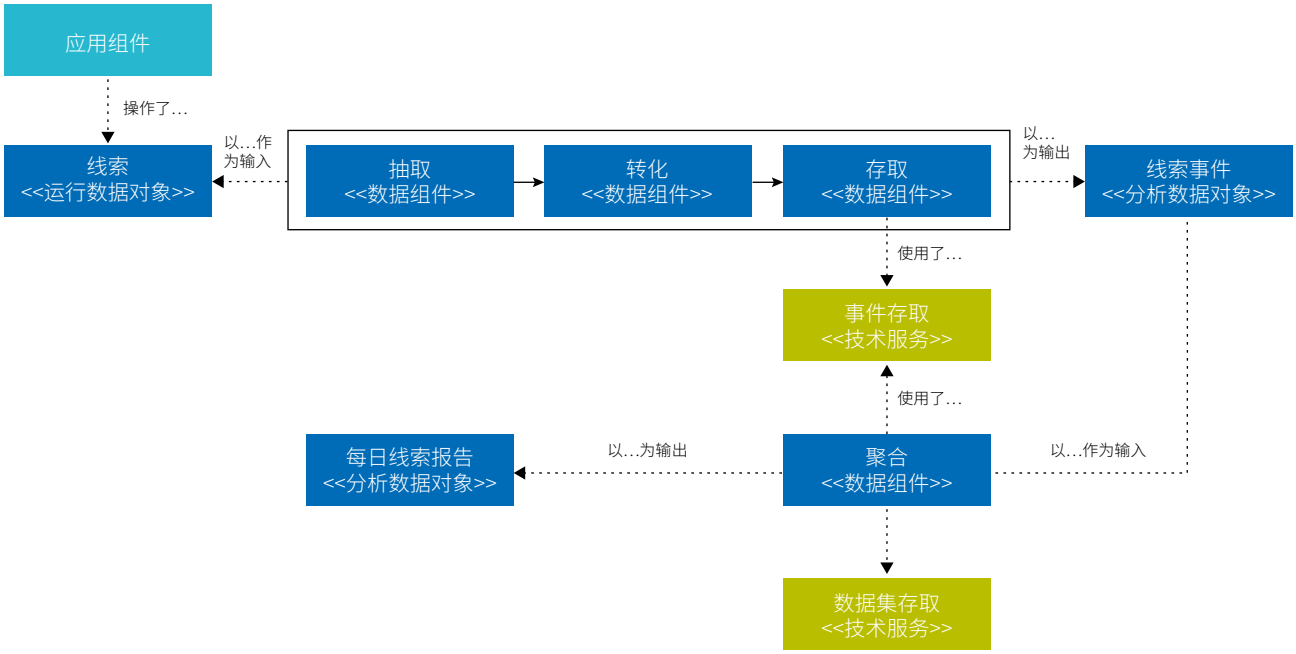
5.2.2.1 数据对象和数据组件建模

数据对象是数据架构的核心模型，是从数据视角对现实世界特征的模拟和抽象。运行类场景的数据对象一般由支撑该场景的应用组件管理，其设计原则往往不适合用于分析。目前主流的处理方式是使用数据流水线将运行类数据从应用组件背后的数据库抽取出来，再进行加工转换，保存到数据仓库 / 数据湖内。

数据组件最主要的作用是为数据加工工序建模，一般对应数据流水线，其将数据对象作为加工的输入和输出。

在建模过程中，往往还会定义数据源，制定数据标准、定义符合分析需要的数据对象的目标模型。在中心化的数据架构中，这项工作一般由专职的数据团队承担，人们认为这是理所当然的，因为只有他们才具备分析类场景的专业技能。这个看法其实不够全面，因为要完成这些工作，除了专业技能外，还需要对业务场景的理解，随着业务场景不断增多，专职的数据团队对于某个具体的业务场景的理解往往弱于负责支持该业务场景的应用开发团队。

并且，由于应用开发团队在职责上不承担分析类场景，其对于数据抽取的支持优先级往往不高，由此，对于由应用内部的数据结构改动可能造成的数据抽取失效不敏感。于是，我们观察到团队之间的合作摩擦和一个不易扩展的数据架构。



(图 5.2-3 一种抽取、加工线索分析数据的建模示例)

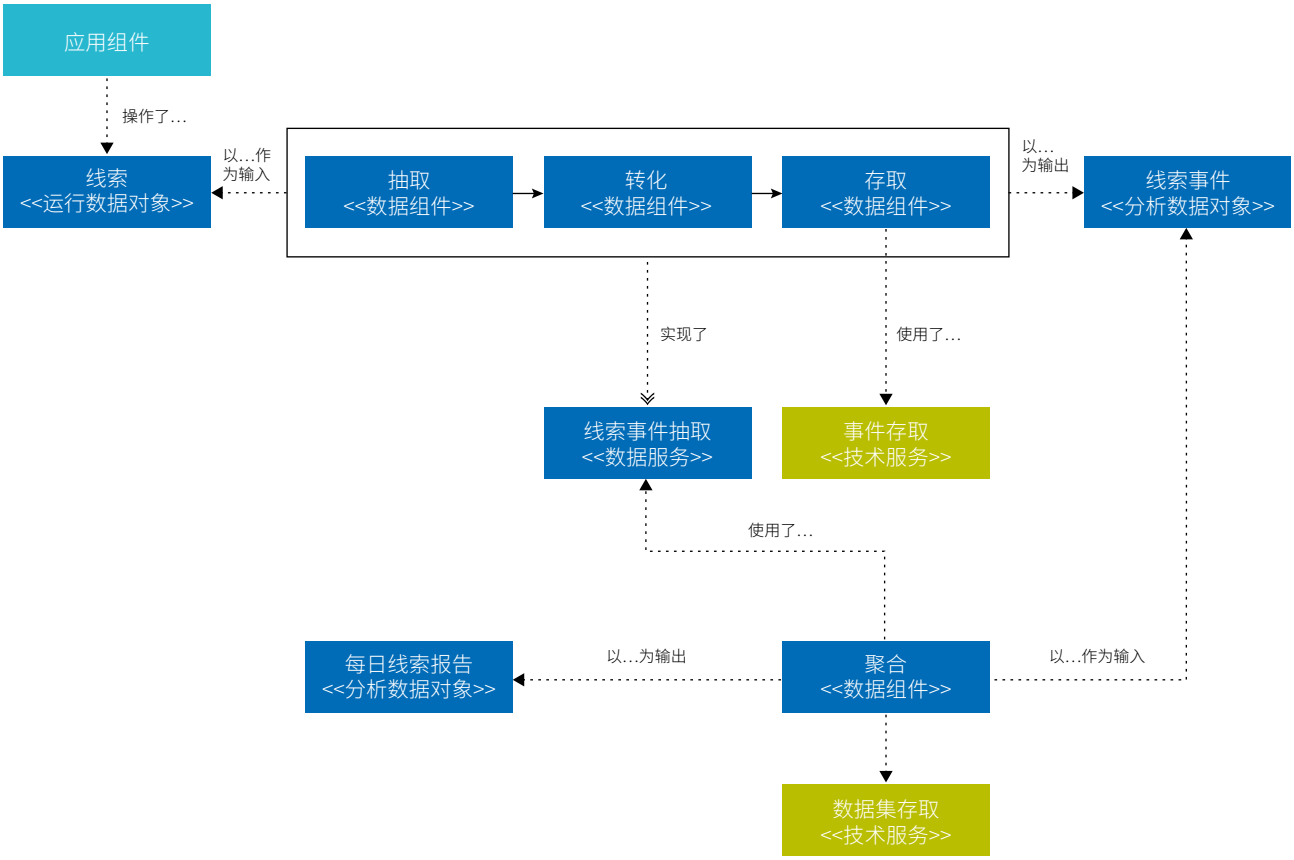
5.2.2.2 数据服务建模

一个反直觉的模式是将分析类场景的职责（或者是其中一部分职责）交还给应用开发团队。在应用架构元模型中，我们会将应用服务视作对外显式定义的一个支撑运行类场景的服务契约，那么是否可以将该理念移植到数据架构呢？即对外显式定义的一个支撑分析类场景的服务契约：数据服务。

例如，可以由应用开发团队承接提供分析类数据原料的职责，提供数据摄取服务（Ingest）。由应用团队与专职的数据团队协作，定义数据原料的格式、质量标准、抽取方案。

通过数据服务，在架构层面，我们显式地定义了职责的边界，作为减少合作摩擦并解放专职数据团队的部分精力的抓手。

另一方面，随着大数据基础设施的服务化，应用开发团队负责相对简单的度量分析也是可想象的空间。如同应用分层一样，分析类数据场景或许也可以分为两层：贴源层和衍生层。



(图 5.2-4 通过数据服务界定职责边界)

贴源层

贴源代表紧靠数据源，某个业务场景自身的业务运营分析需求，其需要的绝大部分数据原料就在支撑这个业务场景的应用中，需求和实现相对稳定。例如，销售线索响应的前置时间趋势，其依赖的主要数据原料是销售线索的跟进事件，它们就在销售线索管理系统这个数据源里。

衍生层

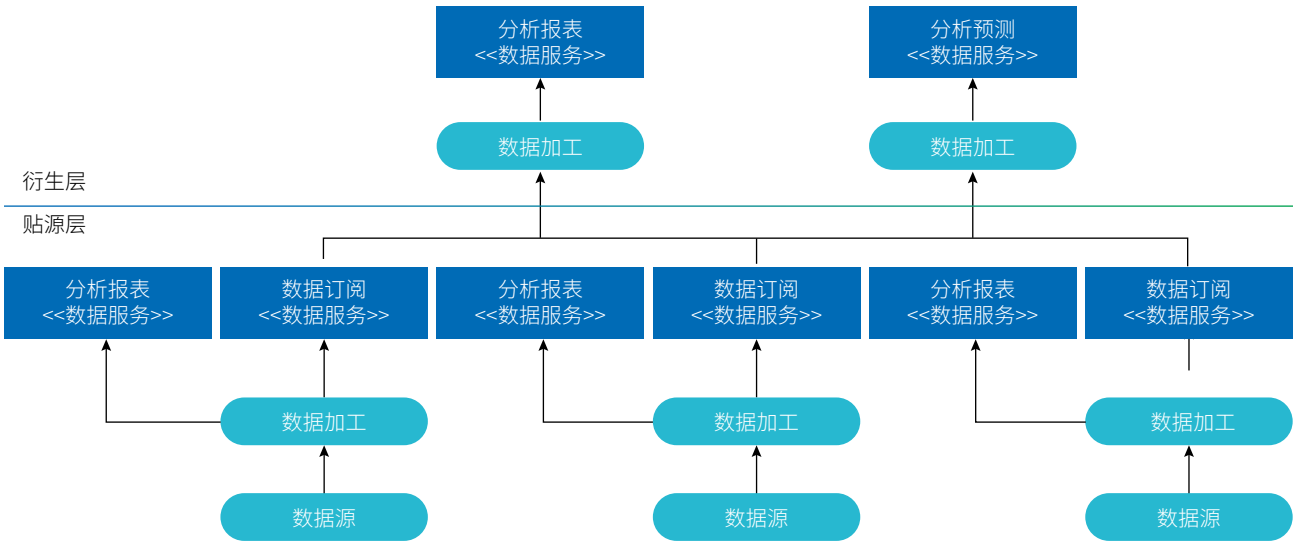
在贴源层之上的是衍生层，这里的分析类场景需要整合多个数据源的数据原料，往往需要经过多次中间处理，实现难度较高，并且需求和实现相对容易变化。例如，整合多个数据源的客户行为数据，为客户打上标签。

那么，将贴源层的度量分析工作交由负责该数据源的应用开发团队，可进一步使得专职大数据团队可以集中精力承担衍生层的工作，从而缓解规模化瓶颈。

这个举措目前的瓶颈是专业知识和工具。因为即使是贴源层，我们仍然建议按照运行类、分析类场景区分数据集的建模方式，分析类数据集往往为了存储时间相关的不可变事实，有更大的数据量，还是需要使用大数据技术栈来存储和加工的。只是前者到后者的加工过程现在由同一个团队负责，减少跨团队协作成本。

所幸的是，在专业知识层面，贴源层的建模相对简单，标准也容易对齐。而工具层面，对大数据基础设施提出了更高的自助服务要求，如果企业选择了云作为解决方案，主流云厂商都提供了托管的大数据产品。

目前对于企业级数据架构尤其是分析类场景的去中心化趋势已经初见端倪，在技术社区逐渐兴起的 Data Mesh(数据网格，参考文献 8) 也逐渐被社区采纳和实践，我们在企业级数据架构框架元模型上的设计也为企业响应这样的趋势提供了基础和弹性。



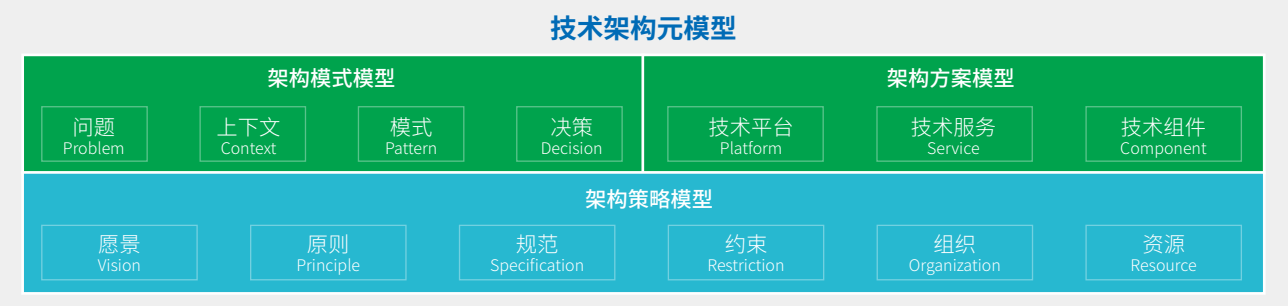
(图 5.2-5 贴源层与衍生层示例)

6

现代企业架构框架 — 技术架构

技术架构是对某一技术问题（需求）解决方案的结构化描述，由构成解决方案的组件结构及之间的交互关系构成。广义上的技术架构是一系列涵盖多类技术问题设计方案的统称，例如部署方案、存储方案、缓存方案、日志方案等等。

企业架构中的技术架构聚焦在对业务、应用、数据等上层架构设计意图的开发实施方案的结构化描述。我们希望结合当前企业数字化建设的主流趋势和新技术成熟普及的大环境，阐述我们对技术架构设计的观察及思考。



(图 6.1-1 企业级技术架构元模型)

6.1 技术架构元模型综述

技术架构元模型是对技术架构组成要素的抽象建模，用来定义用于结构化描述架构设计的模型元素，技术架构元模型的定义需要满足当今企业数字化建设的实际需要。

为了适应当今企业对技术架构的描述需求，我们在经典企业架构框架方法的基础上对技术架构元模型进行了补充扩展，内容主要由架构模式模型、架构方案模型、以及技术策略模型组成。

6.1.1 架构模式元模型

模式分析是快速认识问题本质以及经验复用的有效实践，我们在元模型内容中增加了架构模式模型，引入模式分析视角，对上层架构设计意图、问题进行分析建模，目的是快速、准确定位设计和复用技术方案。

6.1.2 架构方案元模型

架构方案模型是描述技术架构设计的核心元模型，包含三个主要核心元素。基于平台型企业架构技术设计的特征，我们使用了平台、服务、组件这三个层次递进的元素对技术架构进行建模。

6.1.3 架构策略元模型

架构策略模型是为了约束和规范架构设计过程，保证架构设计遵循企业整体的架构设计愿景与需求，符合企业整体的架构设计原则与规范，是对于架构设计过程本身的约束和指导。

需要说明的是，架构策略模型同样适用于业务架构、应用架构和数据架构部分，例如企业级的数据标准就属于架构策略元模型的规范对应的内容。

6.2 技术架构元模型应用

6.2.1 富技术时代如何做好平台型技术架构设计

受益于新技术的涌现和不断成熟，及技术工具的极大丰富，技术架构设计的灵活度和效率都得到了显著提升。

另一方面，在平台型技术架构的设计中，作为多业务条线、多应用、多数据场景落地的技术基座，技术架构设计所需覆盖的规模、应对的复杂度今非昔比。加之“富”技术条件的加持，一个好的技术架构设计的困难度实际上指数级增加。而一直以来，

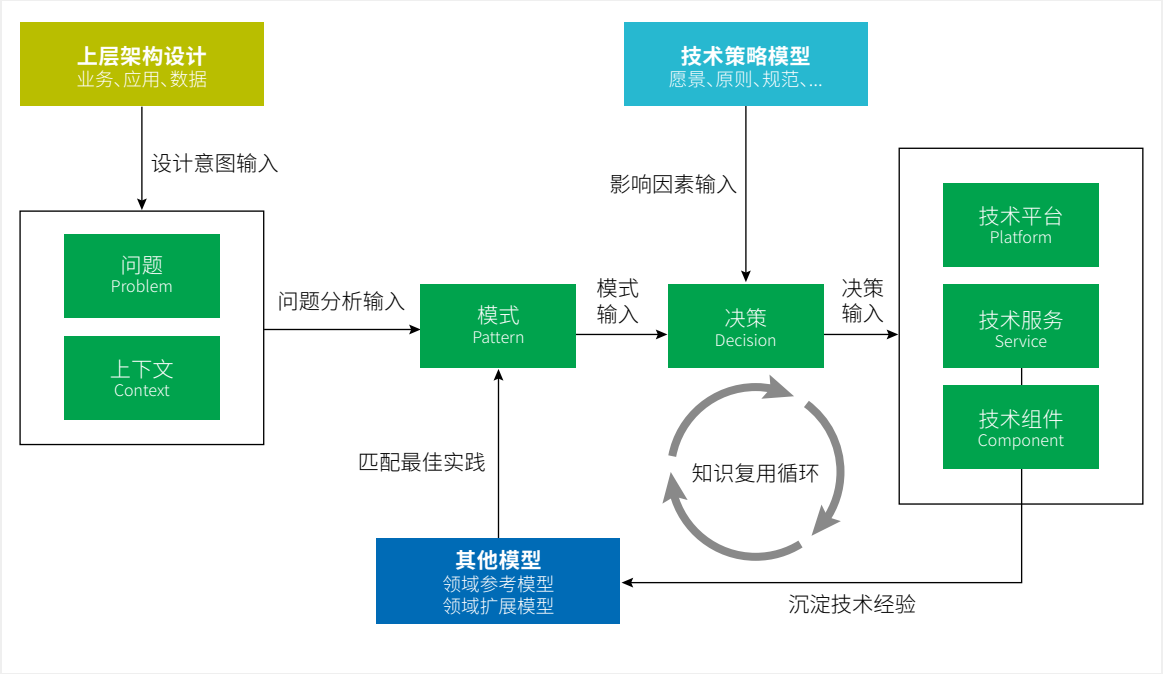
本质上是强依赖架构师的经验和能力的技术架构设计方法和过程，在这样的语境下，一系列挑战和问题再次凸显：

- 对于架构需求把握不足或者没有架构需求的分析意识，过早的进入架构设计，导致系统复杂度变高甚至过度设计，为开发落地带来额外的研发成本
- 架构设计采用的技术和工具过于超前，超出团队成员技术水平，造成落地难度高，新成员上手速度慢，进而对整体进度和实施效果造成影响
- 架构设计过程时间长，完成后团队就不再愿意对设计方案继续调整和迭代，当技术发展变化很快时，设计完成时方案已经过时

因此，富技术时代下，做好平台型技术架构设计的关键是：

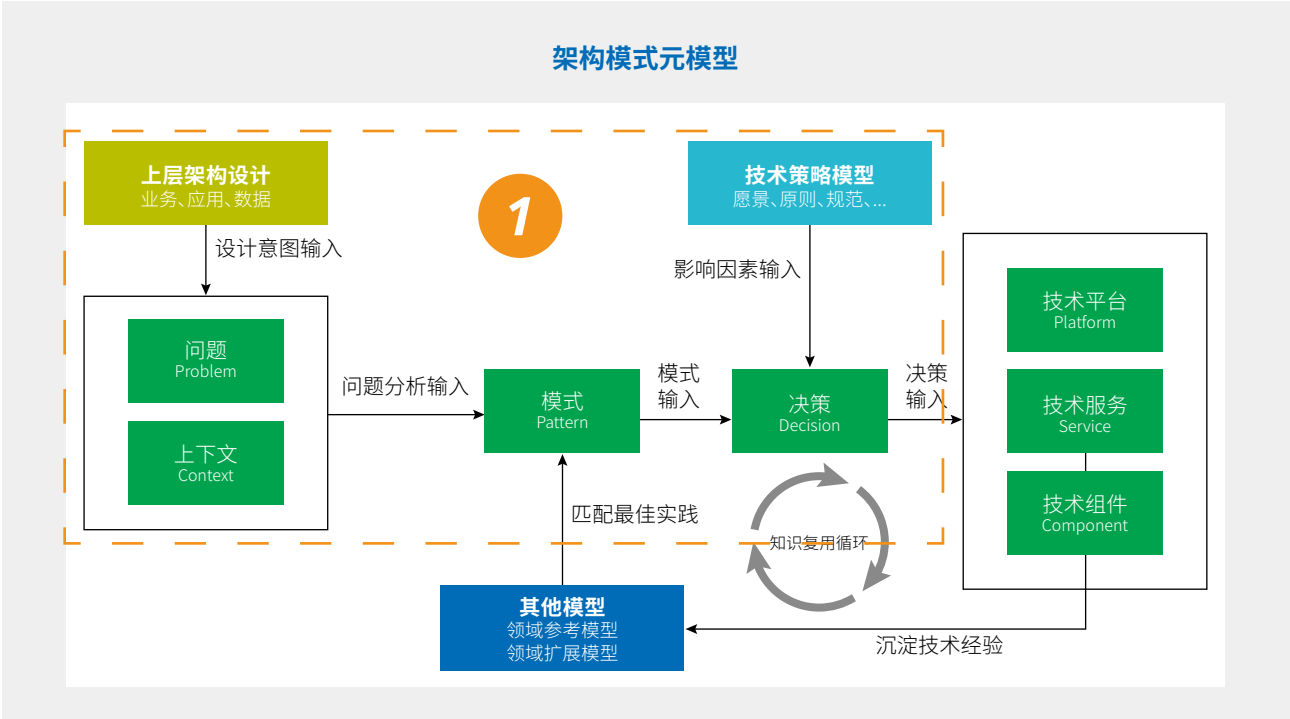
- 系统性的分析架构需求
- 结构化的设计架构方案
- 沉淀可复用的架构经验

技术架构元模型的应用



(图 6.2-2 技术架构元模型的应用)

6.2.2 系统性的分析架构需求



(图 6.2-3 技术架构元模型的应用)

在架构咨询过程中，我们看到很多由不良架构引发的问题现象，分析背后的核心因素时，都指向了一个关键原因，即缺少前期对架构设计需求的系统性分析。当技术团队被问到为什么使用某种设计思路，为什么使用某种技术组件时，得到回答往往跟其自身的主观经验有很大关系。

这带来的重要影响是架构质量与设计者经验密切相关，而经验的传递成本很高，架构决策过程中的信息基本都被丢弃，只留下架构设计结果，导致架构最终难以演进和迭代。因此我们在技术架构元模型中增加了架构模式元模型，引入模式分析的方法对架构的设计过程进行建模。

问题 Problem、上下文 Context

问题和上下文是对上层架构设计输入的分析和解读。

问题描述了架构需求背后要解决的实际问题是什么，例如业务中台中如何保证前台获得一致的服务等级承诺（SLA）。

上下文描述了与问题相关的背景信息，例如问题产生的背景是什么，需要考虑什么样约束条件，期望达到什么样的效果等等。

模式 Pattern

模式是通过对问题和上下文的分析，快速映射到的业界或企业内的最佳实践。

模式是解决某一类问题的方案原理的总结，通过模式技术人员可以快速构成对问题及方案背后原理的理解，在问题不变时，模式具有相对的稳定性，是沉淀技术知识的最佳形式。

决策 Decision

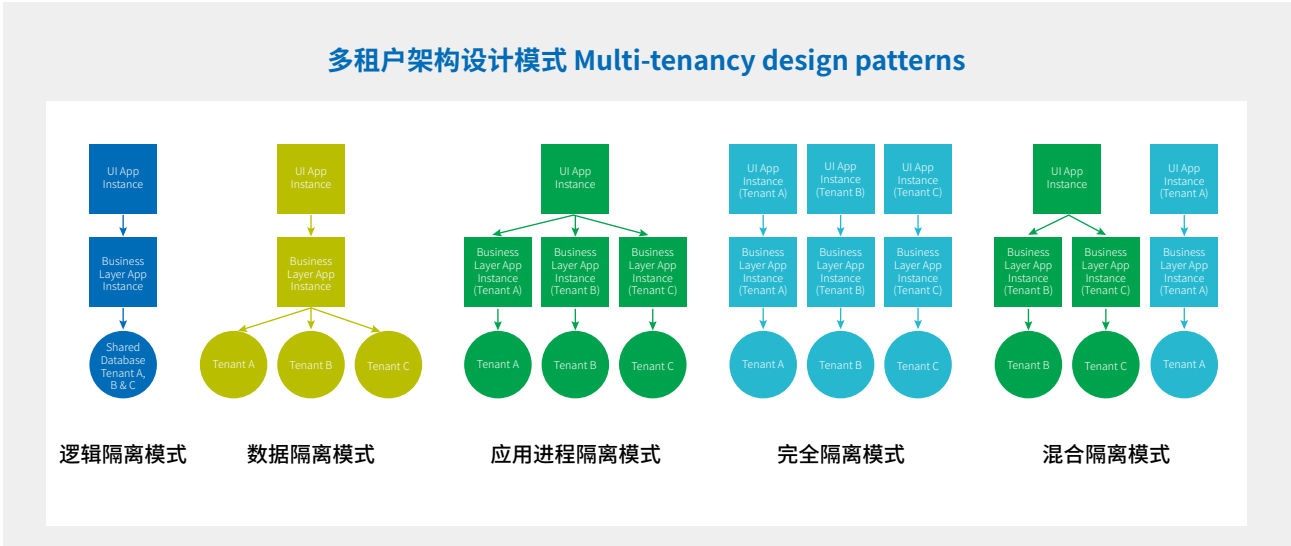
决策描述了在模式的基础上，引入与具体架构方案设计相关的影响因素后，形成的符合满足架构建设需求的技术类决策，决策的描述方式可以是决策树或决策表。

对决策的建模有助于使企业建立起规范的技术决策管理，规范化决策过程及决策内容，是企业构建可演进式的架构治理能力的关键。通常决策的影响因素包括来自顶层设计的 IT 技术战略、架构策略、技术选型、跨功能需求、IT 实施方法等。

实践总结

通常使用问题、上下文对上层设计意图进行系统性的分析后，得到的问题如果准确，那么它在业界往往已经存在成熟的解决方案模式可以参考。架构模式元模型的价值是帮助企业识别和利用已经成熟的最佳实践，提高架构设计质量，降低架构设计成本。

我们以上面提到的中台如何提供一致的服务等级这个问题为例，经过分析，背后的技术问题定义是如何处理接入前台之间的跨功能需求（安全、存储、性能、可靠性等）隔离问题，由此可以快速确定对应的基础架构是多租户（Multi-tenancy）架构。

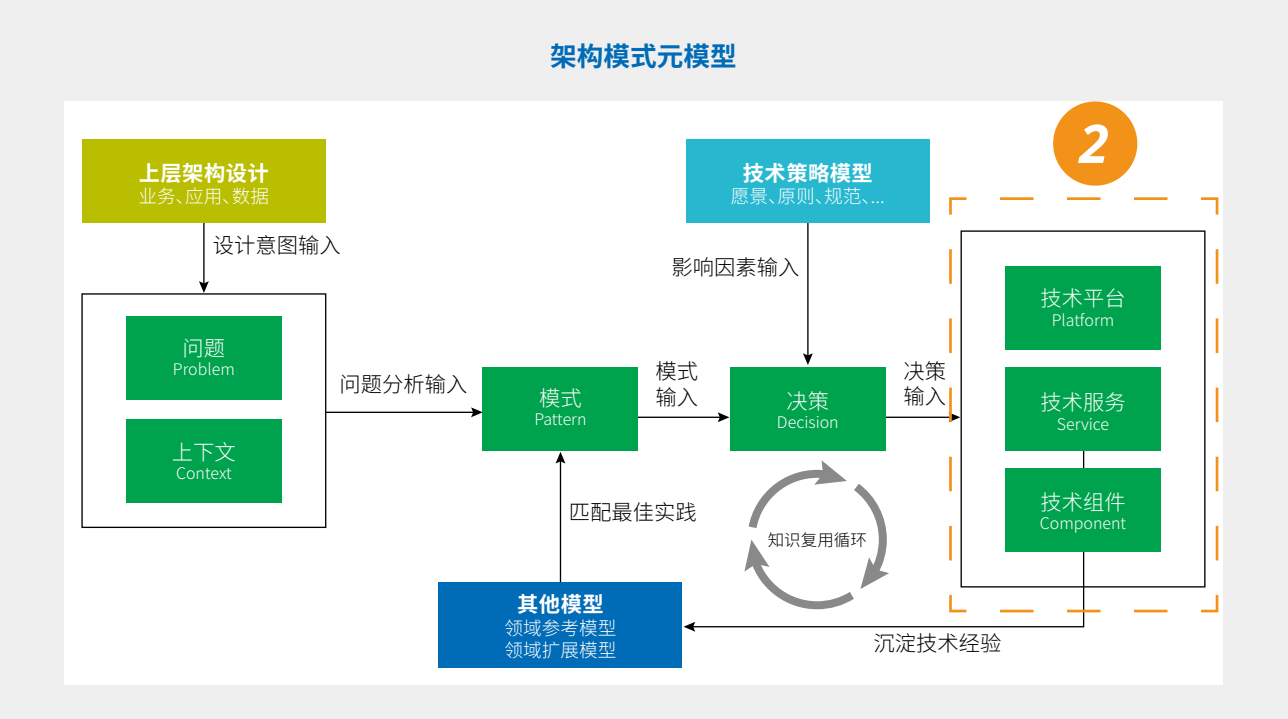


(图 6.2-4 多租户架构设计模式示例)

多租户架构在业界有标准化的成熟模型可以参考，因此我们可以将其作为参考架构，再结合上下文中的需求背景做架构细化，最后引入技术策略模型进行技术选型、实施规划等方面的技术决策，产出最终技术架构方案。

至此我们通过以上四个元模型元素描述了对架构设计过程的建模，实际应用中，每一个元素可以根据企业的架构设计规范，建立对应的参考模型（分类、图示、描述）用以规范架构过程的产出物标准，这里暂不进行展开。

6.2.3 结构化的设计架构方案

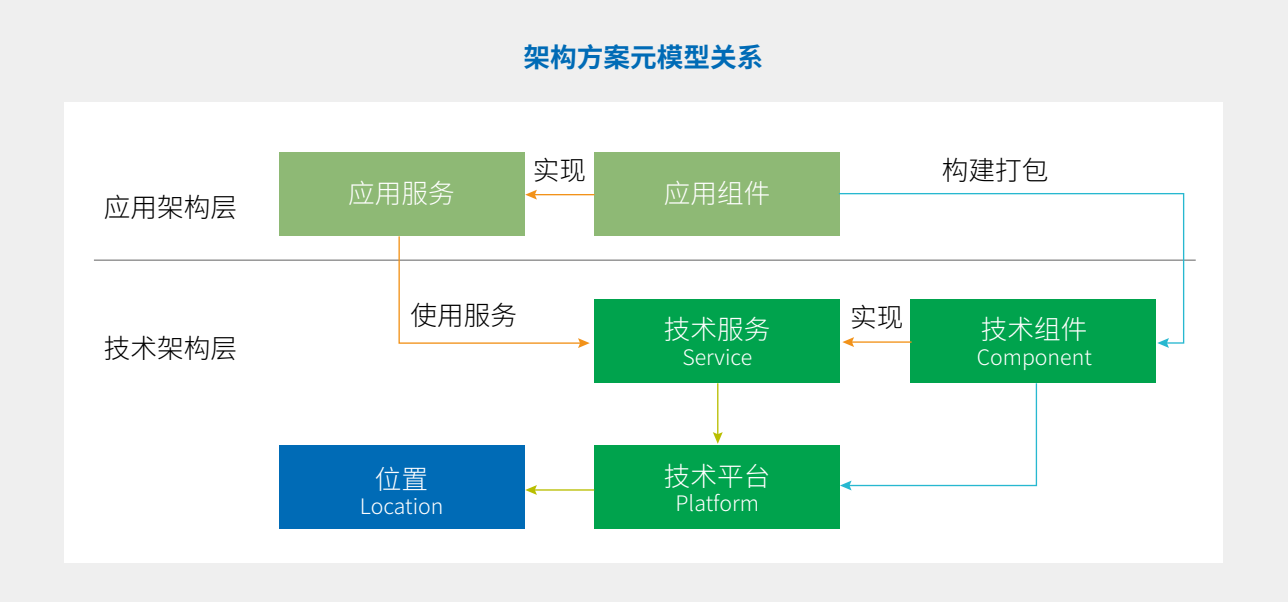


(图 6.2-5 架构方案模型)

在复杂的平台型技术架构中，是否能够对架构元素做准确的识别和直观的描述，直接影响架构设计方案是否被易于理解、使用和管理。在现实世界中，结构化是我们理解、记忆和描述复杂事物的最佳方式，我们希望将其应用在架构设计中来增强架构的表现力，因此我们在设计时对元模型的主要考量因素有：

1. 架构元素的职责明确且易于理解
2. 架构元素的职责之间相互正交
3. 架构元素之间存在清晰可辨的层次关系

相对于经典技术架构元素，我们弱化了逻辑、物理上的分类，使用带有明确职责属性的分类方式定义架构元素，同时我们对元素的职责进行了符合平台化特征的重新定义，最终组成轻量的结构化的描述元模型。



(图 6.2-6 技术架构层与应用架构层元模型关系)

技术组件 Component

技术组件用于描述技术服务的实现，是可部署的物理组件，例如可运行的软件系统或构建打包后的应用组件，技术组件通过架构模式的决策元素，与技术选型进行关联。

与技术组件相关的映射示例		
名称	**类型**	**技术组件**
数据存储	技术服务	MySQL
销售订单服务	应用组件	OrderService JAR
API 网关	技术服务	Zuul

技术服务 Service

技术服务用于描述实现上层架构设计意图所需的技术能力（或功能），例如网关、防火墙、数据存储、缓存等。技术服务属于逻辑模型，作为一种对服务能力的描述，与之相关的 SLA 等跨功能性需求会同时作为其参考描述信息。

技术服务的描述示例（参考 SLA 服务等级描述）	
名称	XXX 网关服务
描述	XXX
职责	XXX
正常运行保证时间	99.99%
响应时间	<= 10ms
故障处理时间	故障等级 1 < 24h，故障等级 2 < 48h
服务提供者	XXX 技术组件 / XXX 云平台

技术服务的价值在于将上层架构中的技术需求与实现相分离，以保证架构设计的稳定性和实施上的灵活性。在技术架构治理中，技术服务是企业 IT 的核

心能力对外的重要展现形式，也是 IT 的核心资产之一，从技术服务的角度实施管理将有助于提升企业整体 IT 服务水平。

技术平台 Platform

技术平台是用于描述由一组技术服务构成，提供解决特定技术领域能力的逻辑模型，它主要用于从更高的层次对技术服务进行管理，简化架构参与者对复杂架构的理解和使用，统一对用户提供一致的 SLA 服务承诺。

下面是技术平台在架构设计中的典型用途：

1. 技术平台作为技术服务的提供者

例如微服务架构通常需要多种技术服务提供支撑，在一些企业内部或技术产品中，将其统一归入微服务平台。

2. 技术平台作为技术组件部署运行的承载者

例如当架构中需要描述部署方案时，技术平台可作为对部署载体的描述，例如提供容器化运行的 PaaS 平台。

3. 技术平台作为外部服务的提供者

例如使用 AWS 等云平台提供的技术服务时，架构设计上即可忽略对技术服务支撑的技术组件的描述。

实践总结

结构化的架构描述有利于企业从多种视角对架构进行管理和治理，实际架构设计过程中，可以由技术服务为切入点展开高阶设计和详细设计。

首先明确上层架构设计意图中对技术能力的依赖是什么，进而定义出技术服务所需提供的 SLA 服务承诺等级，结合架构模式中的决策模型选择技术组件的选型。

对于需要多种技术服务组成而成的能力描述加入技术平台进行统一描述，在高阶设计中以技术平台和技术服务为主描述清楚架构意图和逻辑，在详细架构设计中进而展开技术组件级别的设计。

6.2.4 沉淀可复用的技术知识

技术架构中的复用涉及两个方面：

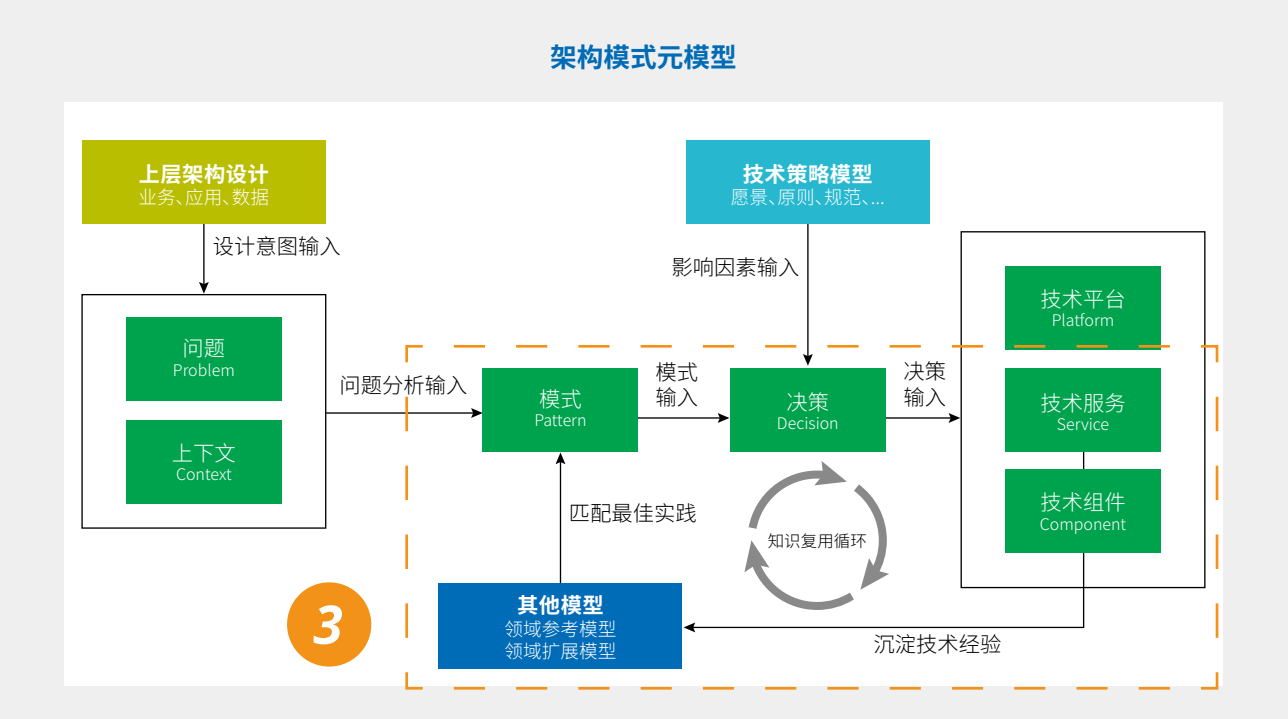
1. 通过技术组件、平台提供可共享的技术服务
2. 架构设计过程中产生的可重复利用的技术知识

技术服务共享在企业中已经通过 IaaS、PaaS 等技术平台得以实现，在最后一个实践中，我们要关注的问题是技术知识的复用。

企业架构设计背后都是成本的投入，从技术管理的角度，管理者希望一次投入可以在更长的时间里产生更多的价值；从技术设计的角度，架构师希望用更短的时间完成更高质量的架构方案。因此很多企业在技术治理策略中，将技术知识作为重要的技术资产进行管理。

下面从技术架构知识管理关注的两个核心问题展开我们对元模型在知识复用上的设计意图：

- 当完成架构设计后，我们从中得到了什么？
- 有什么是可以在以后的设计中重复利用？



(图 6.2-7 复用技术架构设计)

当完成架构设计后，我们从中得到了什么？

通过元模型可以看到，架构设计的产出物包含结果产出物和过程产出物，过程产出物往往在设计过程中作为隐性内容被忽视了，对架构过程的建模的目的之一就是将过程产出物显性的表现出来。

结果产出物的价值主要体现在对建设实施的指导，而过程产出物则代表了对问题、方案的思考分析过程，其价值主要体现在技术知识传递环节，在知识管理领域中 “渔” 比 “鱼” 价值更高。

架构模式元模型中元素同样是对 “经验” 的结构化表达，使仅存储在少部分设计者大脑中的信息得以可视化的展现，便于经验的传递和学习，这也是架构模式元模型的重要作用之一。

有什么是可以在以后的设计中重复利用？

在现实世界中，能够被广泛重复利用的事物都有一个共同的特征，就是在较长的时间里具有很高的稳定性。稳定是可复用的基本前提，复用的价值是从外部变化时而自身可以不变中的获得的。

在技术架构元模型中，架构方案是架构模式在特定场景下的实例化结果，其中特定场景包含上层架构输入的设计意图、影响决策的技术策略等，这些因环境而变的信息是不稳定部分。

架构方案 = 模式 (问题，上下文，决策)

因此在一个架构设计中，问题、上下文、决策都是变量，只有模式是稳定的。

最后我们可以将可复用的技术知识对应到三类架构产出物上：

模式	解决某类问题的最佳实践，只要问题存在，模式就是稳定不变的，不受使用场景的变化而变化，可以结合领域扩展模型对模式细节展开描述。
经验	经验是特定场景中对模式的实例化应用的过程记录，经验可以加快对模式的理解，学习如何结合实际场景应用模式解决问题，经验的内容由架构模式元模型中的问题、上下文、决策三个元素组成，每个元素可以通过定义对应的参考模型展开描述。
方案	最后是架构设计结果，作为案例供后续参考。

7

总结

至此，我们完成了针对现代企业架构框架（MEAF）的四个主要部分的阐述，即企业级业务架构、企业级应用架构、企业级数据架构和企业级技术架构的核心元模型以及元模型的应用场景和建模方法。

在框架的构建过程中，我们希望结合 ThoughtWorks 多年在敏捷精益、企业架构顶层规划与领域驱动设计为特色的 IT 系统落地、以用户为中心的产品化设计等方面的经验与实践，融合成熟的企业架构理论与企业架构框架方法，针对平台型企业架构规划与落地的新背景，为正处于数字化转型浪潮之中，尤其是以平台化架构为原生企业架构选型的企业，总结与提炼一套适配可真正落地的轻量级企业架构框架方法论，以应对现阶段以平台化中台化为代表，从面向业务功能到面向企业核心能力的现代企业架构规划和设计过程中遇到的一些实际问题。

最后，在企业数字化转型中实践越多，我们越知道企业数字化转型的艰难与复杂，也越深感在云原生、平台原生、分布式架构大放异彩，企业不断强调敏捷、创新、产品化、可落地性的今天，需要重新关注企业架构领域以及相应的方法论，并持续结合当前的背景和趋势对于企业架构框架做出相应的演进与适配，使之真正成为真正能帮助企业在现代化架构、技术和趋势的背景下完成数字化转型的重要武器，助力企业数字化转型成功。

8

参考文献

参考文献 1 《国际数据公司（IDC）3 月份的 CXO 月度调研》 国际数据公司

参考文献 2 《趋势洞察：数字加速，在危机时期推动增长的主要技术》 IBM 商业价值研究院

参考文献 3 《从技术走向商业看“中台” 投资机会》 中银国际证券

参考文献 4 《中国行业趋势报告 -2020 年度特别报告》 罗兰贝格

参考文献 5 2019 恒生电子技术开放日

参考文献 6 A Framework for Evaluation of Enterprise World Academy of Science, Babak Darvish Rouhani, Mohd Naz’ri Mahrin, Fatemeh Nikpay, Maryam Khanian Najafabadi, Pourya Nikfard，Engineering and Technology International Journal of Economics and Management Engineering Vol:9, No:1, 2015

参考文献 7 《Domain-Driven Design》 Eric Evans

参考文献 8 《Data Mesh Principles and Logical Architecture》 Zhamak Dehghani

参考文献 9 《TOGAF 标准 9.1 版》 The Open Group, 张新国等译 ,2016

参考文献 10 《Enterprise_architecture_framework》 Wikipedia

作者：

孔磊 首席咨询师

马徐 首席咨询师

王健 首席咨询师

赵志桐 首席咨询师

周宇刚 首席咨询师

（以上排名不分先后，作者均来自 ThoughtWorks 中国 EMPC 业务线）

EMPC 业务团队：

EMPC(Enterprise Modernization, Platform & Cloud，现代化企业、平台及云业务线）作为 ThoughtWorks 全球业务线之一，致力于成为企业数字化转型的全方位合作伙伴。EMPC 围绕以下四个业务主航道，构建领先的市场竞争力：

1. 数字商业模式探索与验证（Digital Business Model Discovery and Validation），旨在帮助面临数字化转型挑战中的客户探索“如何围绕数字化能力构建新的商业模式”。
2. 企业架构设计与演进（Enterprise Architecture Design and Evolution），以中台为支点，帮助企业进行现代数字企业架构的设计与演进，改造遗留系统，加速企业数字化转型的进程。
3. 研发基础设施（Development Infrastructure）通过规划设计一站式研发平台，结合云原生架构，构建高效灵活的研发过程支撑平台，同时采用先进的技术理念和工具帮助客户提升研发效能和架构能力。
4. 云原生转型及演进（Cloud Native Transformation and Evolution）旨在帮助企业充分利用云和云原生技术，围绕云迁移、云原生平台、多云架构及全球基础设施，设计和实施云原生转型蓝图，快速提升业务响应力并重塑 IT 能力，加速数字化进程。

ThoughtWorks®

联系方式：

010-56933000

marketing-china@thoughtworks.com

ThoughtWorks 洞见

ThoughtWorks 商业洞见

ThoughtWorks®