极客时间

极客时间 Java 进阶训练营
第 27 课
分布式系统架构--高并发系统架构设计

KimmKing

Apache Dubbo/ShardingSphere PMC

个人介绍

Apache Dubbo/ShardingSphere PMC

前某集团高级技术总监/阿里架构师/某银行北京研发中心负责人

阿里云 MVP、腾讯 TVP、TGO 会员

10 多年研发管理和架构经验

熟悉海量并发低延迟交易系统的设计实现

目录

1. 到底什么是架构设计

什么是架构（Architecture）

架构（Architecture）一词源于建筑领域，就是建筑的意思，也是体系结构的意思。

维基百科英文版里对 Architecture 的解释是：规划、设计和建造建筑物的过程及产物。

Architect/Architecture ?

软件工程 ~ 建筑工程

- 人的因素
- 标准的因素
- 技术的因素
- 产品的因素，软件边际成本是0

为什么建筑工程如此成熟?

什么是架构

国际标准化组织（International Orgranization for Standardization，ISO）系统和软件工程标准认为，系统的架构是一系列基本概念或者系统在其环境中表现出来的属性，体现在它的元素、关系及设计和发展的原则中。

1.架构过程：在系统整个生命周期中构思、定义、表达、记录、交流，验证合适实现，维护和改进架构的过程，也就是设计过程。

2.架构：一个系统体现在其环境中的元素、关系的基本概念或属性，以及其设计和进化原则。

3.架构描述：表达一个架构的工作产出物（通常指的是各种架构图和设计文档）。

4.架构视图：通过系统的某些关注点的视角，表达一个系统的工作产出物（例如部署视图、开发视图等）。

5.系统：包含了一个或多个进程，硬件，软件，工具与可以满足需求的人的集合。

6.环境：决定了开发、操作、策略和其他影响系统的设置和条件。

什么是架构

总结：软件架构是一个用于指导系统实现的草图，这个草图越详细对于系统实现的指导意义越重大，贯穿于软件的整个生命周期。

在建筑领域，大楼尚未建造前，就已经存在于建筑师的脑海里；同样地，系统开始编写第一行代码之前，就已经存在于软件架构师的心里。至于怎么样把架构草图表达出来呢？我们一般都是采用架构图和设计文档的形式。

架构相关的文档就是，用来描述和交流系统架构的媒介。

架构的一些概念

架构领域长时间发展，积累了很多的概念和基础知识。

例如，我们经常说得软件领域名词还有模式、组件、服务、模块、类库、框架，平台等，

他们跟架构有什么关系和区别呢？

架构的一些概念-模式

模式是表示上下文环境、动机、解决方案三个方面关系的一个规则，每个模式描述了一个在特定上下文环境里不断重复发生的问题的一类解决方案。UML 中给出的解释更通俗易懂：模式是对于普遍问题的普遍解决方案。

我们把一类问题的共性抽象出来，这样就可以用同样的处理办法去解决这些问题，从而形成模式，所以模式是一些经验的总结。

从这个角度来说，软件架构作为一种软件设计过程的指导准则，也是一些经验的积累和问题的抽象，同样也可以作为一种模式。

更一般的，依据于处理问题领域的粒度不同，我们可以把模式分为架构模式（Architectural Pattern）、设计模式（Design Pattern）、实现模式（Implementation Pattern）三个层次。

- 架构模式是最高层次的模式，在软件过程里描述系统的基础结构、子系统划分，确定职责和边界，以及相互作用关系。
- 设计模式是用来处理解决程序设计里具体场景下的问题的解决办法。
- 实现模式是最低层次的具体问题处理办法，例如编码规范、命名规则等。

架构的一些概念-类库

类库（Library）是一组可复用的功能或工具的集合，应用系统通过调用它们从而达到复用功能的目的。

例如，windows 应用开发里的各种静态或动态链接库 DLL 文件，Java 开发里项目里依赖的或者 maven 中央库里的各种 jar 包，都是 Library，比如 apache commons-io、httpclient，log4j 等。

类库根据其所在的语言或平台环境的不同，可以是编译后的二进制执行码或中间码形式（DLL 或 jar），也可以是源代码（PHP、NodeJS 里的类库）。类库的调用关系一般在开发期引入到目标应用的项目中，运行期执行实际调用。

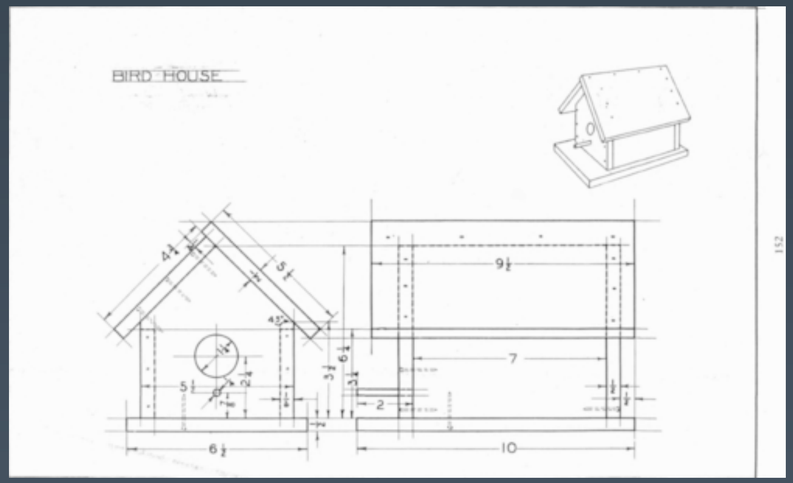
架构的一些概念-框架

框架是基于一组类库或工具，在特定领域里根据一定的规则组合成的、开放性的应用骨架，比如 SSM/SSH 框架，甚至 Dotnetfx、JDK 都算是一种框架。框架具有如下特性：

a）支撑性+扩展性：框架不解决具体的业务功能问题，我们可以在框架的基础上添加各种具体的业务功能、定制特性，从而形成具体的业务应用系统。

b）聚合性+约束性：框架是多种技术点的按照一定规则的聚合体。我们采用了某种框架也就意味着做出了技术选型的取舍。在很多种可能的技术组合里确定了一种具体的实现方式，后续的其他工作都会从这些技术出发，也需要遵循这些规则，所以框架本身影响到研发过程里的方方面面。

在一个具体的框架之上添加一些基本或可复用的功能，这时候就得到一个介于框架和应用之间的结构，我们一般叫脚手架（Scaffold），可以用来快速的实现类似项目。



架构的一些概念-模块

模块（Module）

模块是业务或系统的按照特定维度的一种切分，同时也可以看做是各种功能按照某种分类聚合的一种形式。例如我们的一个电商系统，可以从业务上划分为用户模块，商品模块，订单模块，支付模块，物流模块，售后模块等。另一方面，我们也可以说用户模块聚合了用户注册、用户验证等业务功能。这样，我们在设计和开发过程中，就可以按照模块的维度去组织，比如每个模块新建一个源码的子项目（subproject）、打包成一个单独的jar包，也可以放到一个项目里用不同的 package 名称来区分等。模块一般是系统在较大粒度上的解耦切分，仅次于系统或子系统的级别。

架构的一些概念-组件

组件（Component）

组件是一组可以复用的业务功能的集合，包含一些对象及其行为。组件可以直接被用做业务系统的组成部分，粒度一般小于模块，也是一种功能的聚合形式。比如日志组件、权限组件等。根据组件的形式、行为和用途的不同，我们又可以延伸一些概念：

- 构件（Composite）：具有层次组合关系的多个组件组合形成的复杂组件形式。比如 RCP 里一个Window 里左边嵌套一个 TreeView 组件、右边添加一个 GridView 组件，这样就形成了一个 Composite 构件。
- 部件（Widget）：部件主要是有 UI 界面的构件，比如 Windows 7或 Mac 系统自带的桌面天气小部件等。
- 插件（Plugin）：系统运行期间可以即插即用、随时停用卸载的组件，一般有确定的生命周期，比如 google Atom 编辑器的各种插件、OSGi 中的 bundle、Eclipse 插件（本质上也是 OSGi 的 bundle）等。

架构的一些概念-服务

服务（Service）

结构化信息标准促进组织（Organization for the Advancement of Structured Information Standards，简称 OASIS，XML 和 WebService 规范就是这个组织提出的）把服务定义为：

一种允许访问一个或多个功能的机制，其中访问需要使用规定的接口，并且与服务描述中指定的约束和策略一致（a mechanism to enable access to one or more capabilities, where the access is provided using a prescribed interface and is exercised consistent with constraints and policies as specified by the service description）。

概括来说，服务是一组对外提供业务处理能力的功能，服务需要使用明确的接口方式（比如 WebService 或 Rest 等），服务描述里应该包括约束和策略（比如参数、返回值，使用什么通讯协议和数据格式等）。

架构的一些概念-平台

平台（Platform）

一般来说，平台是一个领域或方向上的生态系统，是很多解决方案的集大成者，提供了很多的服务、接口、规范、标准、功能、工具等。例如 J2EE 平台，包含了企业级应用开发里的各种基于 Java 语言和 JVM 虚拟机运行时的技术能力。

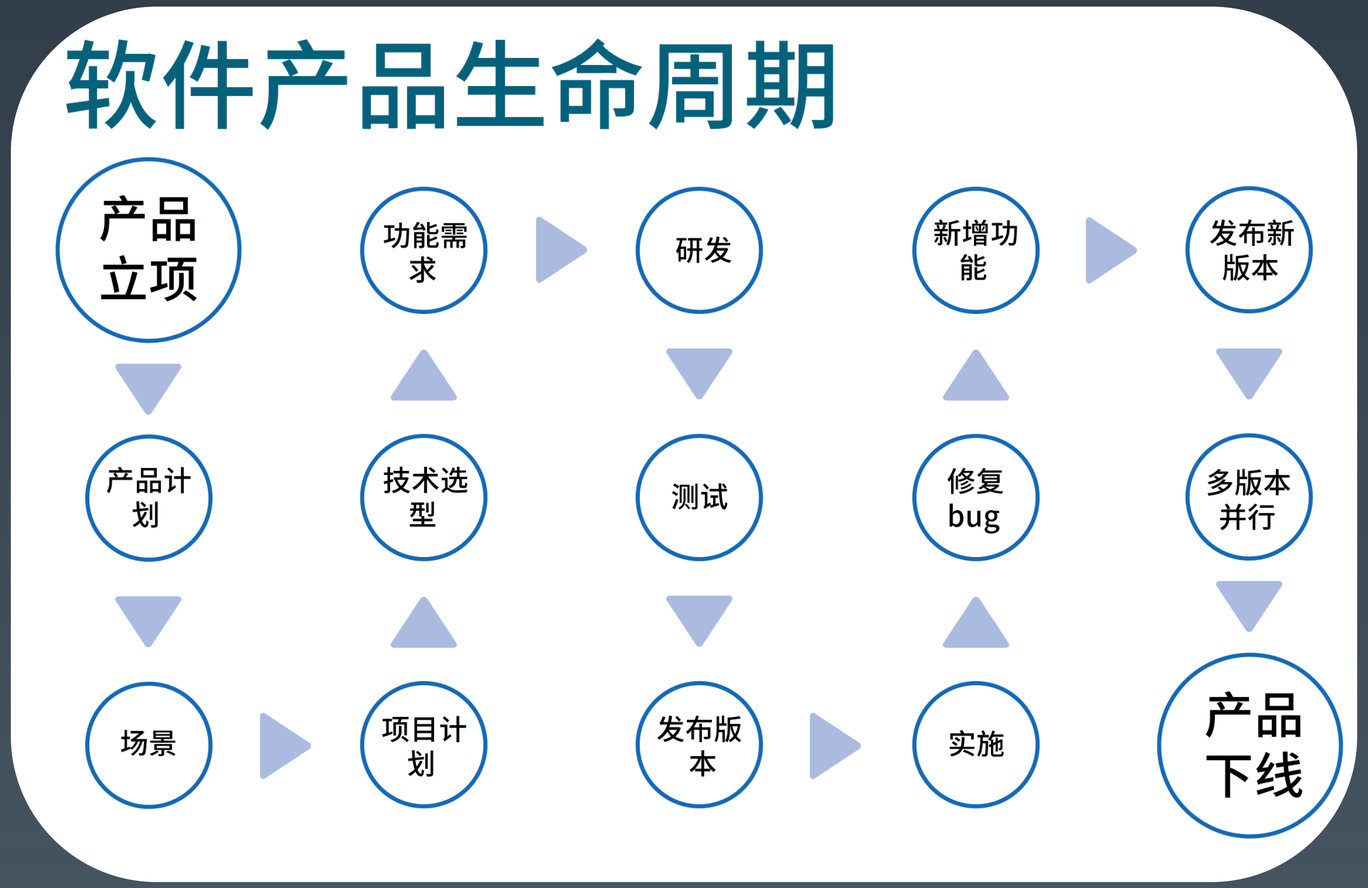
- 库是工具箱。
- 框架是一套通用的解决方案。
- 架构是高度抽象的需求，是系统中的不变量。
- 平台是所有可能做的事的集合。

事实上，服务、平台、架构这几个概念这几年已经被泛化了，什么地方都可以滥用这几个词，随便一个系统都可以说自己是大数据平台，XX 业务平台，XXX 服务化架构。

一个 SOA 的真实笑话。

从软件生命周期看架构

软件生命周期可以概况为3个大的周期：设计期（包括立项、计划、技术选型与方案等）、实现期（包括开发、测试、发布、实施等）、运行期（或维护期，包括修复 bug、新增功能、多版本维护等）。



从软件生命周期看架构

1、设计期，软件作为一个成品还不存在，所以我们可以称之为概念形态。此时架构师、产品经理或需求分析师等人员利用自己的经验能力，对系统的业务需求进行分析、拆解、抽象，形成业务文档和技术文档，以及技术验证代码等。这个阶段，架构设计工作是重中之重，其中包括：

1. 系统分拆，如何把系统拆解为不同的子系统、模块、业务单元；
2. 技术选型，使用什么样的基础技术框架或脚手架；
3. 技术验证，确定核心技术难点如何解决，检验能否满足期望指标；
4. 接口规范，系统的内部不同部分以何种形式确定接口契约和数据通信；
5. 集成方式，系统与外部其他业务系统如何进行集成；
6. 技术规范，如何规范开发、测试、部署和运维的技术标准性；
7. 部署方案，系统如何进行物理部署，需要多少机器、什么配置，对网络有什么要求；
8. 运维方案，系统如何进行技术性运维，如何日常监控、预警报警；

总结一下：业务为要，架构先行（包括业务架构和技术架构）。

从软件生命周期看架构

2、实现期，这个阶段主要是编码与测试，准备部署上线，是软件从代码到最终的生产系统的过程，我们可以称之为代码形态。此阶段需要考虑的技术类工作包括：

1. 确保各项技术规范和技术指标的执行落地，保障高质量的代码；
2. 指导研发人员和解决各类技术问题，提升研发团队效率；
3. 制定测试的技术性方案和基准，自动化、性能、安全等；
4. 配合准备部署环境，运维实施方案落地等；

实现期的主要任务是大量软件工程师根据设计期的设计编码。大量的人员，大家背景不同，知识储备不同，编程水平和习惯不同，努力程度不同，如何能够让所有工程师都既能够按数量保障项目进度，又能够按质量保障软件品质呢？秘诀就在于：技术标准的精确统一，系统部件的良好拆分，此外最好有适合于此类项目的脚手架，随时能解决各位技术难点问题的救火队。系统部件的良好拆分，保障了任务是可以拆散成一个个的小单元，分发给不同的开发者。技术标准的精确统一，可以实现不同个体的产出物最大程度的一致性。总结为，标准统一，快速开发。

从软件生命周期看架构

3、运行期，这个阶段系统上线、验收通过，已经初步稳定，然后进入维护阶段，成为了设计期架构设计草图的一个可用实例，我们可以称之为实例形态。此时需要考虑：

1. 发布上线相关基础性工作，包括是否使用持续集成（CI)、自动化发布等技术；
2. 运维基础性工作，自动化运维，监控等相关技术；

进入维护期以后，软件系统日常通过内外部用户反馈的问题和改进意见，不时需要修改代码发布补丁版本或者调整数据以满足用户需要。经过较长一段时间，业务模式和内外部环境发生了比较大的变化以后，系统简单的补丁维护可能就不能满足用户需求，需要大范围的添加新功能和修改旧功能的逻辑流程，此时就可以成立专门的团队，重复上面的步骤，基于现有系统重新做一些改造性的设计重构（设计期），再编码实现（实现期），最终再发布一个较大的版本（运行期）。总结为，基础设施，监控运维。

在软件的整个生命周期里，架构师或架构组是一个项目或者产品线的技术负责人，再大一点的组织，比如公司或研发中心级别层面也许还有架构部，架构师、架构组、架构部在不同层面对自己工作涉及到的所有技术问题负责。

其实上面罗列的这些工作汇总一下，再加上技术规划与执行落地、技术人才的选拔与培养等，可以作为项目组架构师或者研发团队架构组的工作职责。

2. 系统架构的演化发展

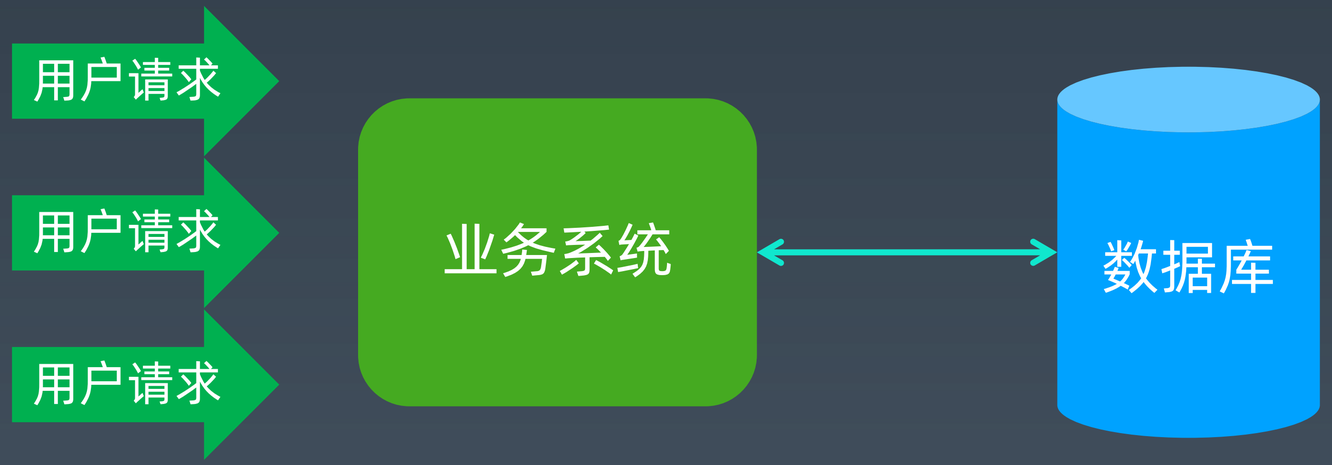
架构发展演化

从软件架构的整体风格来说，大致目前有如下几个阶段。

架构发展演化--以电商为例

1.0 单体版本~ asp/jsp/php 为代表，

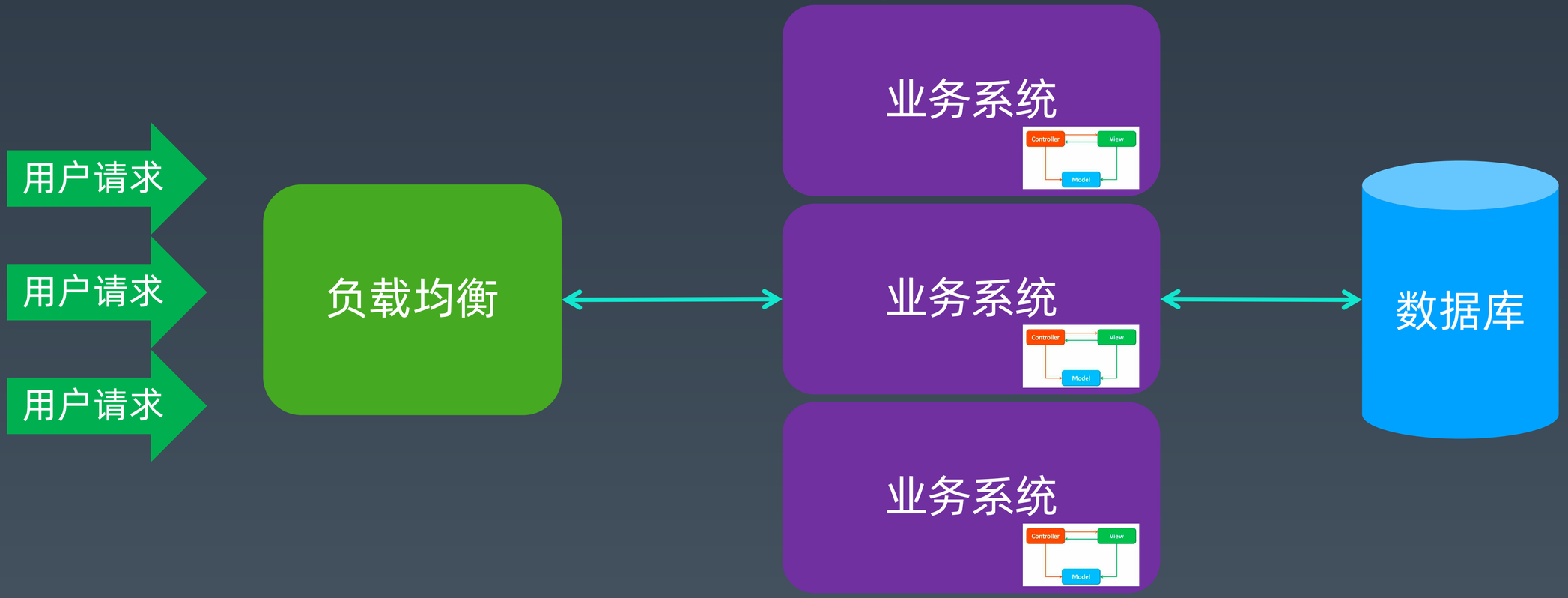
一般以提升硬件性能来提升系统处理能力



架构发展演化--以电商为例

2.0 垂直架构 ~ 关注于大规模 Java 应用开发，以分层为软件层面的设计原则，

大规模的集群部署方式：涉及负载均衡，反向代理，Spring/ORM 等框架。



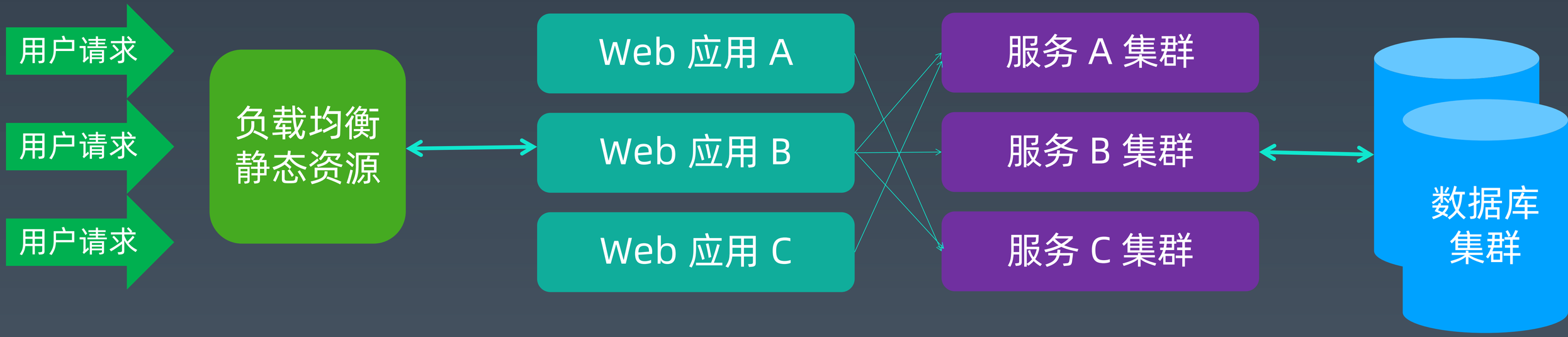
架构发展演化--以电商为例

3.0 SOA/分布式服务化版本，以 RPC 与前后端分离为代表。

引入数据库中间件和复制技术，解决数据库高可用和容量。

引入缓存，解决热点读问题。

引入全文检索，解决海量的复杂条件查询问题。



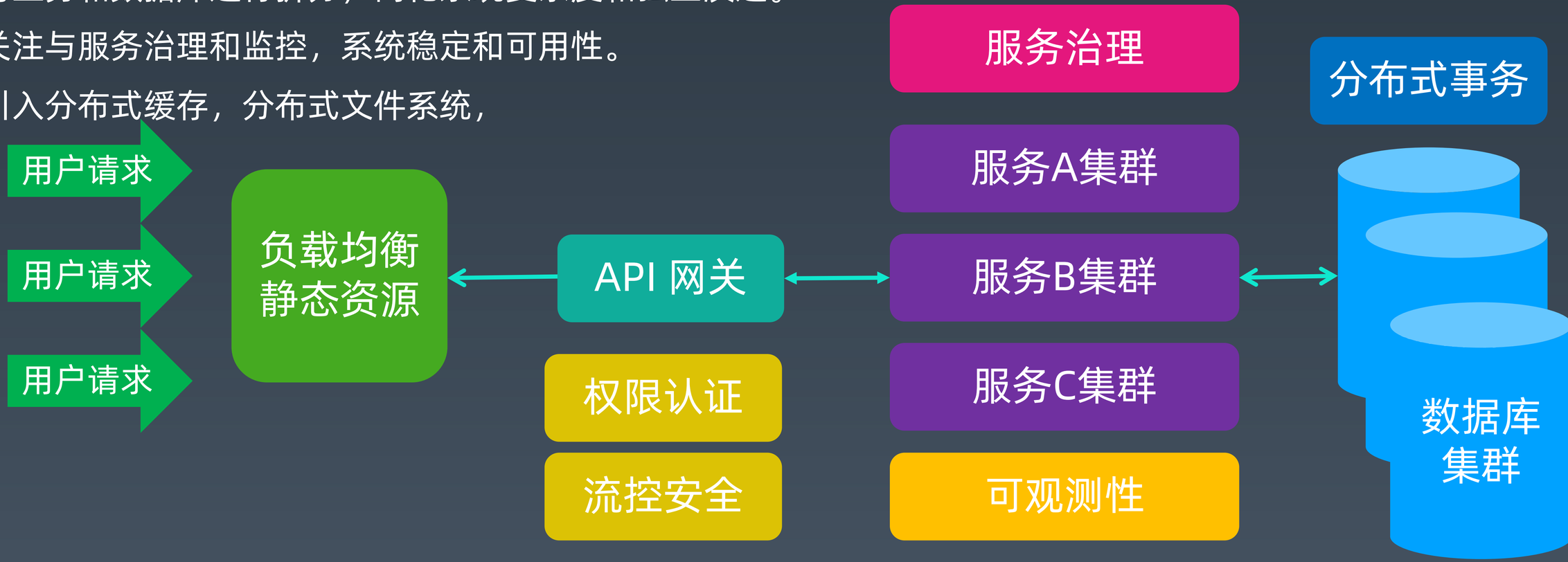
架构发展演化--以电商为例

4.0 微服务版本：大规模分布式服务化+容器化+治理化。

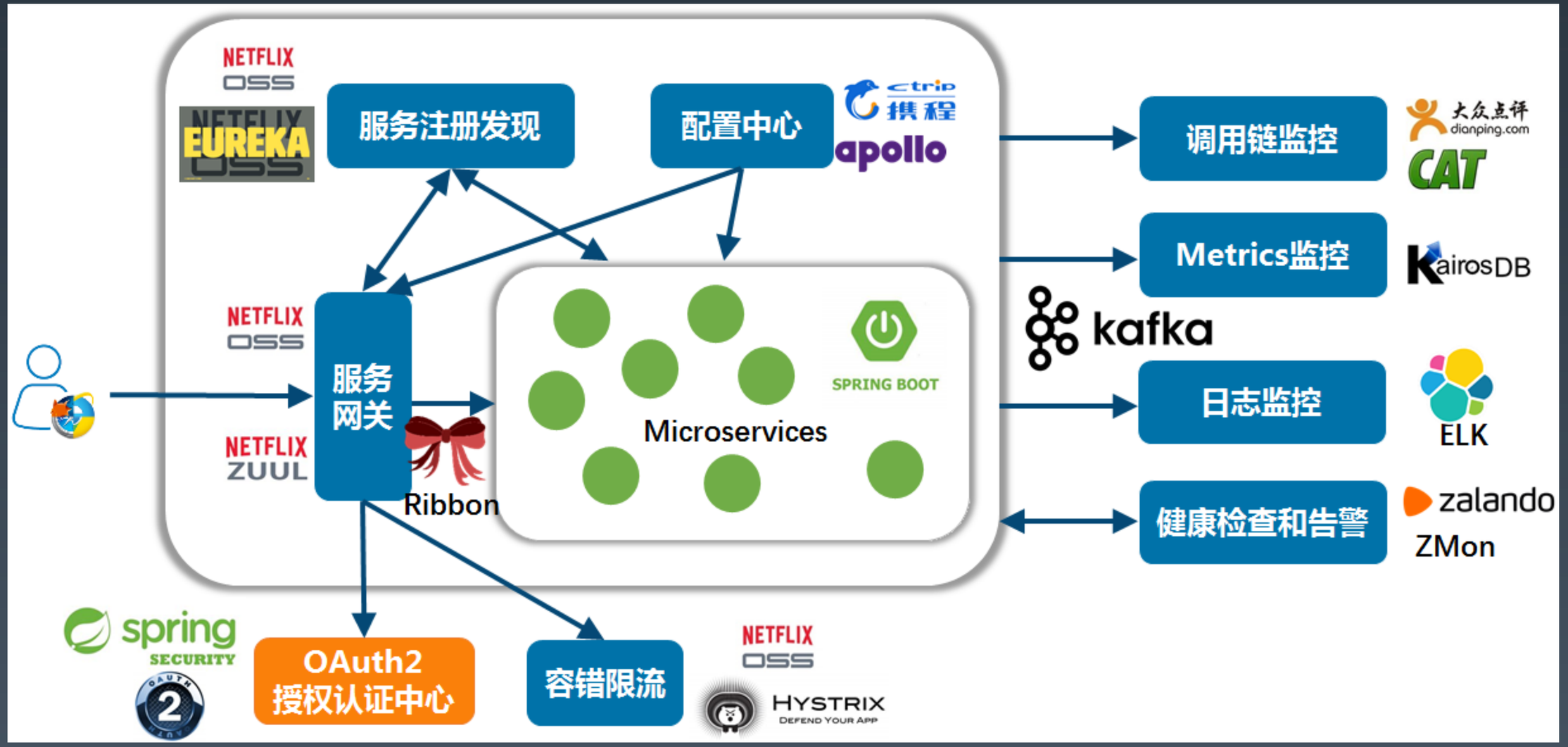
对业务和数据库进行拆分，简化系统复杂度和独立演进。

关注与服务治理和监控，系统稳定和可用性。

引入分布式缓存，分布式文件系统，



架构发展演化--以电商为例



3. 架构设计形式与方法

架构的形式：以文档和代码呈现

我们一般说的架构既包括架构的设计过程，也包括设计的产出物，一般可以包括各类设计文档、设计图，也可以包括一些技术验证代码、Demo或者其他相关程序。文档的目的在于准确记录我们的思维产物，在软件尚未实现时，作为指导蓝图，尽量精确的描述清楚软件。

在软件已经实现以后，部署运行的软件实例和代码只能说明软件目前是什么状态的，却无法告诉我们这个软件系统中间的决策信息。

一个软件系统的长期稳定发展，必然需要一个可靠的、随着软件本身的维护不断同步更新的文档作为每次变更的出发点。这样，我们可以随时沿着架构相关的文档逆流而上，了解这个软件系统从整体到具体的设计思路。

同时文档作为结项或交接的一部分，也是整个软件项目的产出物的一部分，成为公司IT资产的有机组成部分。

架构的形式：以文档和代码呈现

文档是设计的载体，代码是系统功能实现的载体，技术和业务最终都有很大一部分体现在代码里。

广义上来说，代码和代码里的注释，都可以认为是文档的一部分。社区有一种观点：良好结构的、可读性强的代码，是最好的“文档”。

那么怎么才能写出好的代码呢？关键在于两个词：经验、重构。

从代码里，我们可以很直观的了解到，程序做了什么、不能做到什么，以及能做到什么程度，以及其与相关的文档（包括业务文档和技术文档）是否一致。但是代码不适合作为唯一的“文档”，只有代码没有其他文档，就像是一部只有结尾没有开头和过程的电影。我们只能了解到这个系统的一个现在时间点的切面影像。

如何用文档说清楚架构设计呢？

为了清晰的描述软件架构设计，我们就引入了一个概念：架构视图。什么是架构视图呢？Philippe Kruchten 在其著作《Rational 统一过程引论》中写道：

一个架构视图是对于从某一视角或某一点上看到的系统所做的简化描述，描述中涵盖了系统的某一特定方面，而省略了于此方面无关的实体。

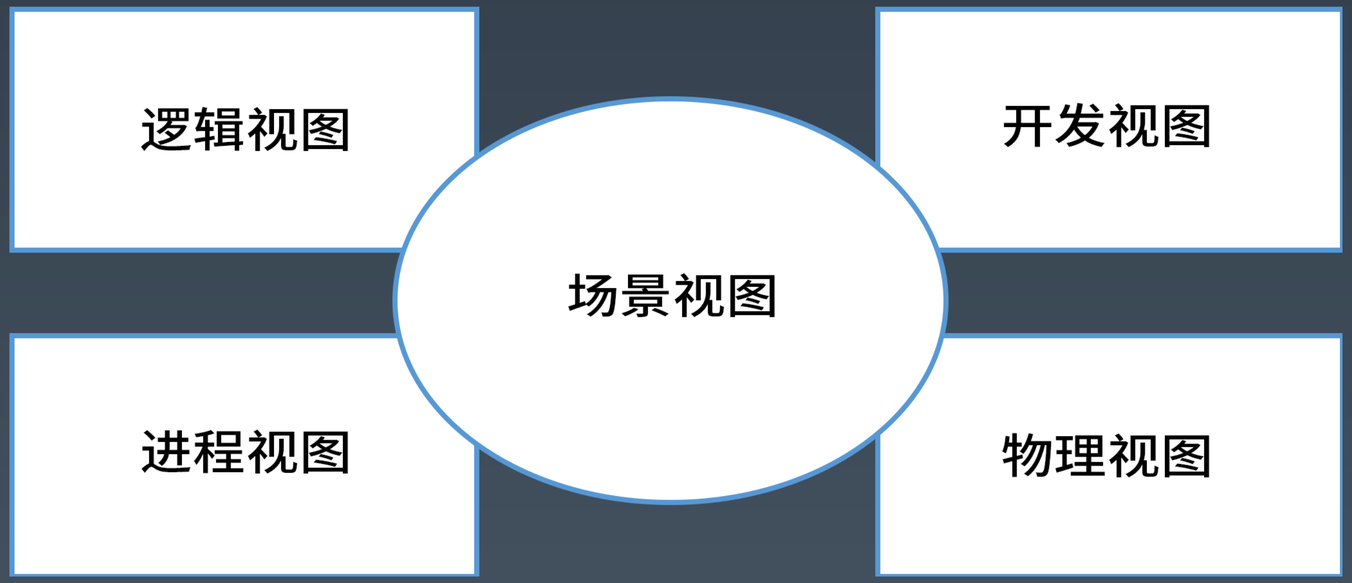
也就是说我们可以使用从不同视角分别描述同一个架构设计，最后把这多个视角的设计综合到一起，就是这个系统的完整架构设计了。这些不同角度的设计文档，也成为了我们理解一个系统的基本依据。

一个新的问题是：哪些视角可以作为全面描述一个系统架构的最核心视图呢？1995年，Philippe Kruchten在《IEEE Software》上发表了论文《4+1架构视图模型（The 4+1 View Model of Architecture）》，正式提出了使用场景视图、逻辑视图、开发视图、进程视图、物理视图五个方面来描述架构设计，引起了业界的极大关注，并最终被后来隶属于 IBM 的 Rational 软件公司统一软件开发过程方法论（Rational Unified Process，简称 RUP）所采纳。

如何用文档说清楚架构设计呢？

在4+1视图模型中，不同架构视图承载不同的架构设计决策，支持不同的目标和用途。

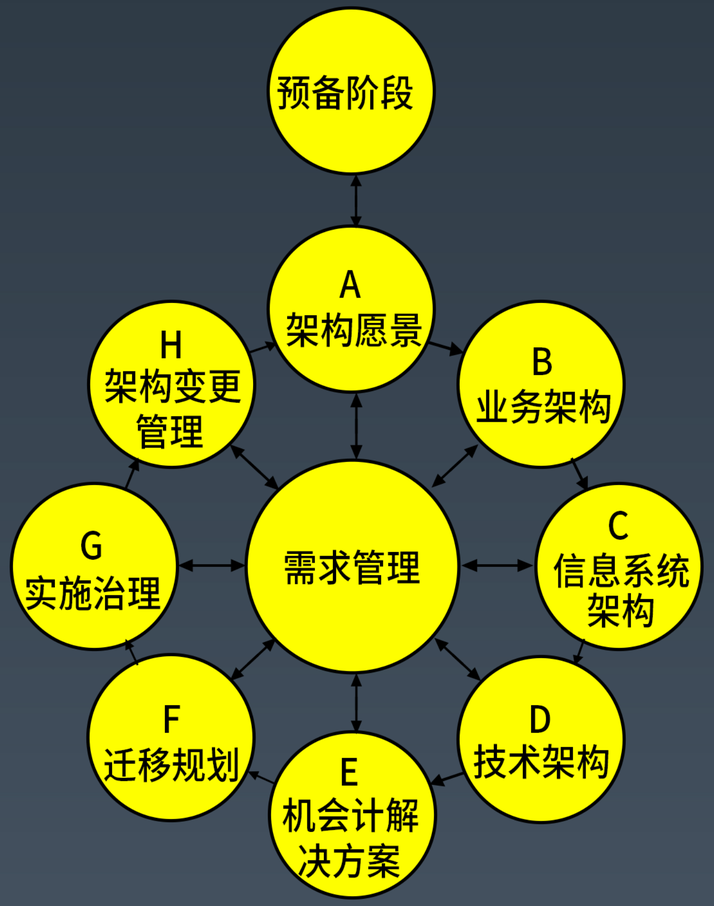
概括来说，我们可以从场景视图的功能需求、逻辑视图的对象与交互、进程视图的进程与通信、开发视图的项目开发组织结构、物理视图的网络与机器部署结构等这五个方面来描述一个系统的架构设计，并形成文档、设计图等设计输出物，用于指导后续的软件实现、测试、部署与维护等过程。

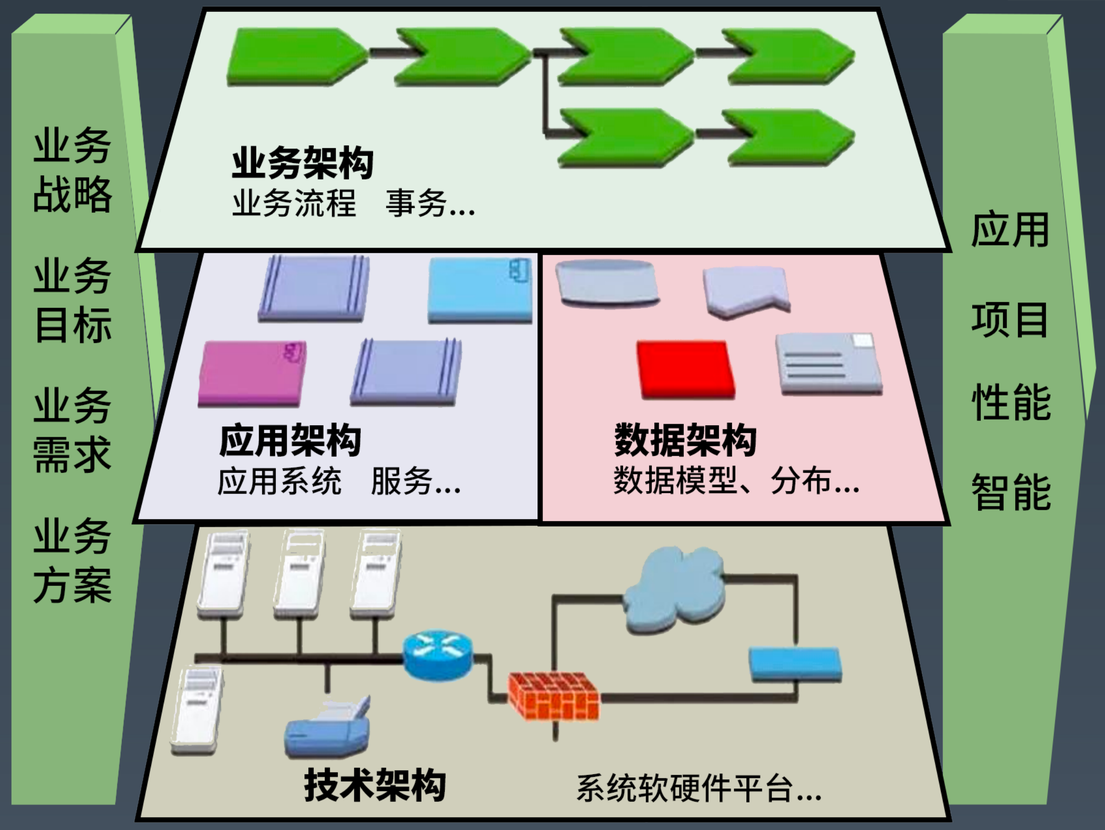


如何用文档说清楚架构设计呢？

1. 用例视图（Use Cases View）：也叫场景视图，关注最终用户需求，为整个技术架构的上线文环境，通常用 UML 用例图和活动图描述。
2. 逻辑视图（Logical view）：主要是整个系统的抽象结构表述，关注系统提供最终用户的功能，不涉及具体的编译即输出和部署，通常在 UML 中用类图，交互图，时序图来表述，类似与我们采用OOA 的对象模型。
3. 开发视图（Development View）：描述软件在开发环境下的静态组织，从程序实现人员的角度透视系统，也叫做实现视图（Implementation View）。开发视图关注程序包，不仅包括要编写的源程序，还包括可以直接使用的第三方 SDK 和现成框架、类库，以及开发的系统将运行于其上的系统软件或中间件，在 UML 中用组件图，包图来表述。开发视图和逻辑视图之间可能存在一定的映射关系：比如逻辑层一般会映射到多个程序包等。
4. 进程视图（Process view）：进程视图关注系统动态运行时，主要是进程以及相关的并发、同步、通信等问题。进程视图和开发视图的关系：开发视图一般偏重程序包在编译时期的静态依赖关系，而这些程序运行起来之后会表现为对象、线程、进程，进程视图比较关注的正是这些运行时单元的交互问题，在 UML 中通常用活动图表述。
5. 物理视图（Physical view ）：物理视图通常也叫做部署视图（Deployment View），是从系统工程师解读系统，关注软件的物流拓扑结，以及如何部署机器和网络来配合软件系统的可靠性、可伸缩性等要求。物理视图和处理视图的关系：处理视图特别关注目标程序的动态执行情况，而物理视图重视目标程序的静态位置问题；物理视图是综合考虑软件系统和整个 IT 系统相互影响的架构视图。

更为系统化的架构方法论-TOGAF





The TOGAF® Standard — Version 9.2

4. 架构的一些实践经验

架构设计服务于业务

正如19世纪的伟大建筑师路易斯•沙利文（Louis Sullivan）倡导的建筑设计著名格言：“功能决定形式（Form follows function）”，软件架构首先是要服务于业务功能的。

设计一栋大楼不管美不美观、大气不大气，首先需要考虑的是这栋大楼是做什么用得，是要开一个百货公司、还是一个跨国集团的总部大楼、还是当地市政府的办公大楼。

同理，架构首先也需要对业务负责。而业务也并非都是一成不变的，随着市场环境的变化、用户习惯的变化、竞争格局的变化，业务形态也一直在顺应环境而改变，架构设计也需要考虑系统在未来的一段时间内支撑这种调整。并且在满足了业务需求和一定的前瞻性的基础上，综合考虑成本、周期、效率、速度、风险等因素。

架构影响研发团队的组织形式

业务拆分的方法和技术框架的选择必然会影响到研发团队的组织形式。

业务拆分的越细致，越有利于我们更好的对项目的各项指标量化计算，更精确的估计工时和成本，从而指导我们每个小组应该分配多少资源，使用什么样的协同和任务确认形式。

反过来，研发组织的结构和成熟度也会对我们最终所采取的技术架构产生重要的影响。比如一个由3个初级程序员组成的创业小团队不适于采取特别复杂且小众的开发框架。相反地，利用快速开发框架或脚手架把产品设计迅速实现应该是团队的核心诉求，所以某种全栈类的全家桶解决方案才可能是最适合的技术选择。

架构存在于每一个系统

每一个已经实现并运行的系统，都是特定架构设计的载体。有些系统对应的架构，有详细的设计文档来描述；有些系统的设计文档，残缺不全，甚至还因为在系统的发展变化的同时，文档没有更新，导致设计文档与实际系统不符；有些系统干脆就没有设计文档。但是这些系统，都是基于一定的架构来创建的。

每种架构方式，每个具体系统内所体现的架构设计，都是可以被工程师们理解，进而提炼出来一些架构思想和设计原则，这些思想和原则就是这种架构方式的风格。

依据这些风格，我们可以将各种架构方式，进行分门别类，从而进一步讨论每种架构风格的特点。例如在实现期的代码形式中，系统由各个相似的类库作为组件构成，在运行期这些组件又同时在同一个进程中，这时我们可以认为这是一种“组件式单体架构风格”。

架构需要警惕过渡设计

矫枉过正，过犹不及。

过渡设计，是一个特别常见的问题。

比如在做架构设计的时候，想当然把问题想复杂了，然后做了过于复杂化的方案。

做技术选型的时候，因为自己熟悉和顺手，选了看起来“高大上”，但是实际上不适合的技术或者组件。

架构需要不断的发展演进

随着计算机软硬件的不断发展，软件架构思想也在不断的发展变化。同时软件为其提供业务处理和服务能力的每个具体行业领域也在不断发展变化，业务处理流程、参与角色、业务形式不断的推陈出新。

这就要求我们在系统架构设计时，保持终身学习的精神，持续吸收新思想新知识，保持贴近一线业务群体，随时因地制宜，调整架构设计，采取最适合当下场景的解决方案。

同时对于存量的旧系统的维护与改造，很多时候无法一次达到目标，可以考虑循序渐进，设定几个大的里程碑，逐步推进，最终实现比较理想的架构设计。

多年以来多很多老旧系统改造的经历，对我影响最深的就是，循序渐进，终达目标，特别是具体的设计实践中，思路和方法，比结论更重要。一个正确的结论在别处可能就是错的，但是思路和方法确实可以复用的。只有通过不断的迭代更新自己的知识和思想，才能一直做出来最适合的架构设计。

第 27 课总结回顾

什么是架构设计

架构的演化发展

架构的形式方法

架构的一些经验

第 27 课作业实践

1、（选做）思考一下自己负责的系统，或者做过的系统，能否描述清楚其架构。

2、（选做）考虑一下，如果让你做一个针对双十一，某东某宝半价抢100个IPhone的活动系统，你该如何考虑，从什么地方入手。

3、（选做）可以自行学习以下参考书的一两本。

推荐架构书籍：

①《软件架构》Mourad Chabane Oussalah

②《架构实战-软件架构设计的过程》Peter EeLes

③《软件系统架构-使用视点和视角与利益相关者合作》Nick Rozanski

④《企业IT架构转型之道》

⑤《大型网站技术架构演进与性能优化》

⑥《银行信息系统架构》

⑦《商业银行分布式架构实践》