一线架构人员需要了解的内容

MORE ABOUT ARCHITECTURE

吕林 naturelin@126.com

总线

框架

设计模式

模型

工作流

SOA

架构

设计

拓扑

组件

需求

规约

分层

系统

模块

UML

问题：软件架构到底是什么？

架构的起源

- 常说软件架构思想起源于建筑行业
- 结构化设计技术是软件架构技术的源头

软件架构的发展史

- 从时间角度看
  - 基础研究阶段（1985年~1994年）
  - 概念确定阶段（1992年~1996年）
- 从技术角度看
  - 面向对象的编程
  - 面向组件的编程
  - 企业级系统框架
  - 面向服务的架构
  - 云计算

软件架构发展史-基础研究阶段

- 60年代-80年代，已经使用一些图线的方式描述系统架构（结构）了，但系统化的、风格通用的、严谨而且是结构化可重用的软件系统设计方法并没有被总结出来。
- 80年代中后期，一些基本思想和概念被提出来，包括：信息隐藏、封装、抽象等。但这些基础概念不足以解决软件系统设计中遇到的其他典型问题，如：如何解决模块之间的依赖性？如何使设计有利于设计和代码的维护和扩展？

软件架构发展史-概念确立阶段

- 90年代开始，架构基本概念和模型被确立。
- 五个标志：
  - 架构描述语言的发展
  - 初步的架构表述及分析规则的制定
  - 架构元素及架构风格的分类研究
  - 架构的评估方法
  - 可借鉴的架构视角（例如:4 + 1视角）
- 概念确立阶段的最后的一个重要实践总结，“架构视角（Architecture View）”概念。

软件架构发展史-技术背景

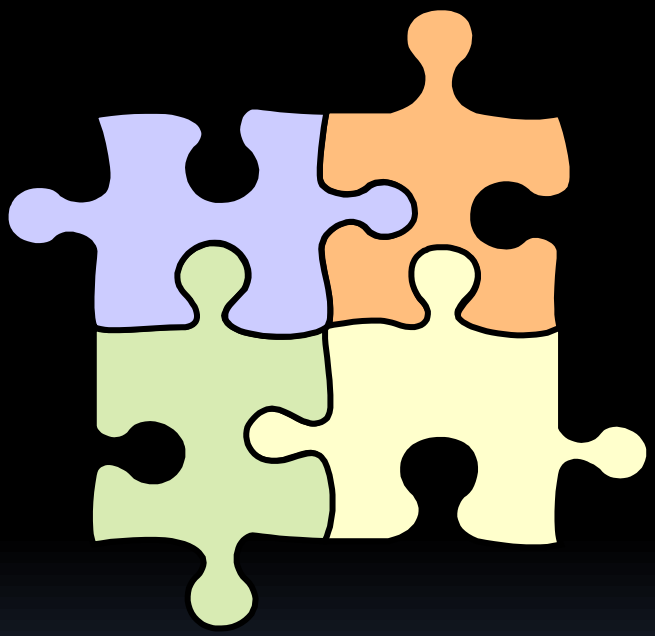
- 软件危机（堪比经济危机）
  - 软件开发费用和进度失控。费用超支、进度拖延的情况屡屡发生。有时为了赶进度或压成本不得不采取一些权宜之计，这样又往往严重损害了软件产品的质量。
  - 生产出来的软件难以维护。很多程序缺乏相应的文档资料，程序中的错误难以定位，难以改正，有时改正了已有的错误又引入新的错误。随着软件的社会拥有量越来越大，维护占用了大量人力、物力和财力。
  - 软件的可靠性差。尽管耗费了大量的人力物力，而系统的正确性却越来越难以保证，出错率大大增加，由于软件错误而造成的损失十分惊人。

软件架构发展史-奠定基础

- 软件工程的出现
  - 1968 年北大西洋公约组织的计算机科学家在联邦德国召开国际会议，第一次讨论软件危机问题，并正式提出“软件工程（Software Engineering）”一词，从此一门新兴的工程学科——软件工程学——为研究和克服软件危机应运而生。
- 结构化编程技术的出现
  - 作为软件工程重要的组成部分，结构化设计方法得到了极大的发展。

软件架构发展史-技术起源

- 结构化设计技术是软件架构技术的起源。
- 软件模块化技术是软件架构技术中最早讨论的技术之一。



软件架构发展史-面向对象的编程

- 面向对象的编程（Object-Oriented Programming）
- 起源于：挪威人Kristen Nygard于1962年发明的Simula语言，美国人Alan Kay于1970年发明了SmaUTalk语言，并首次使用OOP术语。
- 进步：更真实地反映客观世界。提出了设计模式的概念，促进架构的发展。
- 问题：粒度太小，难以驾驭复杂性高的大型系统。

软件架构发展史-面向组件的编程

- 面向组件编程（Component Oriented Programming，COP），是对面向对象的编程的补充，帮助实现更加优秀的软件结构。
- 进步：能够隔离变化，合理的划分系统。而框架的意义就在于定义一个组织组件的方式。
- 问题：DLL Hell。

软件架构发展史-企业级开发框架

- Java
  - Sun公司于1995年推出了Java平台.
  - 在1999年末，Sun提出了Java平台企业版（J2EE），该规范被应用在主要的IT提供商以构建稳健的应用系统框架，如IBM、Oracle和BEA等。
- .NET
  - 2000年，为了应对来自Internet和J2EE的威胁，Microsoft提出了.NET战略。
  - .Net技术的全面推进，统一了Microsoft的不同技术理念和平台。.Net为WebService提供了原生的解决方案，并且成为提升不同应用和系统之间互操作性的标准。

软件架构发展史-面向服务的架构

- 面向服务的架构（Service Oriented Architecture，SOA）是一个组件模型，它将应用程序的不同功能单元（称为服务）通过这些服务之间定义良好的接口和契约联系起来。
- 进步：面向业务的松耦合，使得企业可以按照模块化的方式来添加新服务或更新现有服务，以解决新的业务需要，提供选择从而可以通过不同的渠道提供服务，并可以把企业现有的或已有的应用作为服务，从而保护了现有的IT基础建设投资。

软件架构发展史-云计算

- 云计算（cloud computing）是指IT基础设施的交付和使用模式，指通过网络以按需、易扩展的方式获得所需的资源（硬件、平台、软件）。提供资源的网络被称为“云”。“云”中的资源在使用者看来是可以无限扩展的，并且可以随时获取，按需使用，随时扩展，按使用付费。
- 3个层次：基础设施云、内容云、信息云
- 3个实现模型：私有云、混合云、公有云
- 3种服务模式：IaaS，PaaS，SaaS

软件架构发展史-？

- 社会一体化架构

软件架构发展现状

- 目前，软件体系结构尚处在迅速发展之中。关于软件体系结构的研究工作主要在国外展开的，国内到目前为止对于软件体系结构的研究尚处在起步阶段。软件体系结构在国内未引起人们广泛注意的原因主要有两点：
  - （1）软件体系结构从表面上看起来是一个老话题，似乎没有新东西。
  - （2）与国外相比，国内对大型和超大型复杂软件系统开发的经历相对较少，对软件危机的灾难性体会没有国外深刻，因而对软件体系结构研究的重要性和必要性的认识还不很充分。

到底什么是软件架构？

- 为何做架构？
- 谁来做架构？
- 如何做架构？

为何做架构？

- 开发一个软件系统的需要？
- 领导的指示？
- 公司技术的积累？
- 对原有系统的重构？

为了解决某个问题！

- 软件架构就是一个解决方案。
- 软件架构设计是在规划一套针对某个问题的解决方案的过程。

谁来做架构？

- 架构师
  - 架构师是软件行业中一种新兴职业，工作职责是在一个软件项目开发过程中，将客户的需求转换为规范的开发计划及文本，并制定这个项目的总体架构，指导整个开发团队完成这个计划。架构师的主要任务不是从事具体的软件程序的编写，而是从事更高层次的开发构架工作。他必须对开发技术非常了解，并且需要有良好的组织管理能力。可以这样说，一个架构师工作的好坏决定了整个软件开发项目的成败。

架构师——有N多种

- 企业架构师
- 解决方案架构师
- 平台架构师
- 软件架构师
- 系统架构师
- 前端架构师
- 网络架构师
- 业务架构师
- ……

如何做架构？

- 准备工作
  - 架构设计团队都OK了吗?
  - 架构设计的工作领导小组都OK了吗?
  - 架构的工作流程都清楚了吗？
- 开始架构设计…

软件架构设计的工作流程

- 当明确了要解决的问题之后，我们往往还是不知如何入手？
- 架构设计的几大问题：
  - 架构设计，如何起步？
  - 将系统划分成模块，如何做更合理？
  - 前期需求没能理清，才影响了架构设计！
  - 非功能需求很重要，但是该如何设计呢？

有问题才有解决方案

- 架构设计的整体思路是：
  - **需求进，架构出。**
- 基于这个整体思路，提出我们的架构解决方案：
  - 方法体系是大趋势
  - 质疑驱动的架构设计
  - 多阶段和多视图
  - 有效的实践方法

1.方法体系

- 架构涵盖太多的内容和因素，单一的方法已经不能解决架构的问题。适当的引入多种方法，并加以有效的结合，才能解决架构方面的问题。
- 方法体系包括：
  - 三个阶段
  - 一个贯穿环节
- 通过3各阶段和1个贯穿环节来覆盖“需求进，架构出”的架构设计思路。

2.质疑驱动的架构设计

- 架构设计的源动力：需求。
- 需求清楚了，架构就清楚了吗？
- 架构设计中的核心要素：人！
  - 通过人的思维，将需求有节奏地引入架构设计一波又一波的思维活动中来。
  - 通过“质疑”引入架构的质量属性，以及约束。

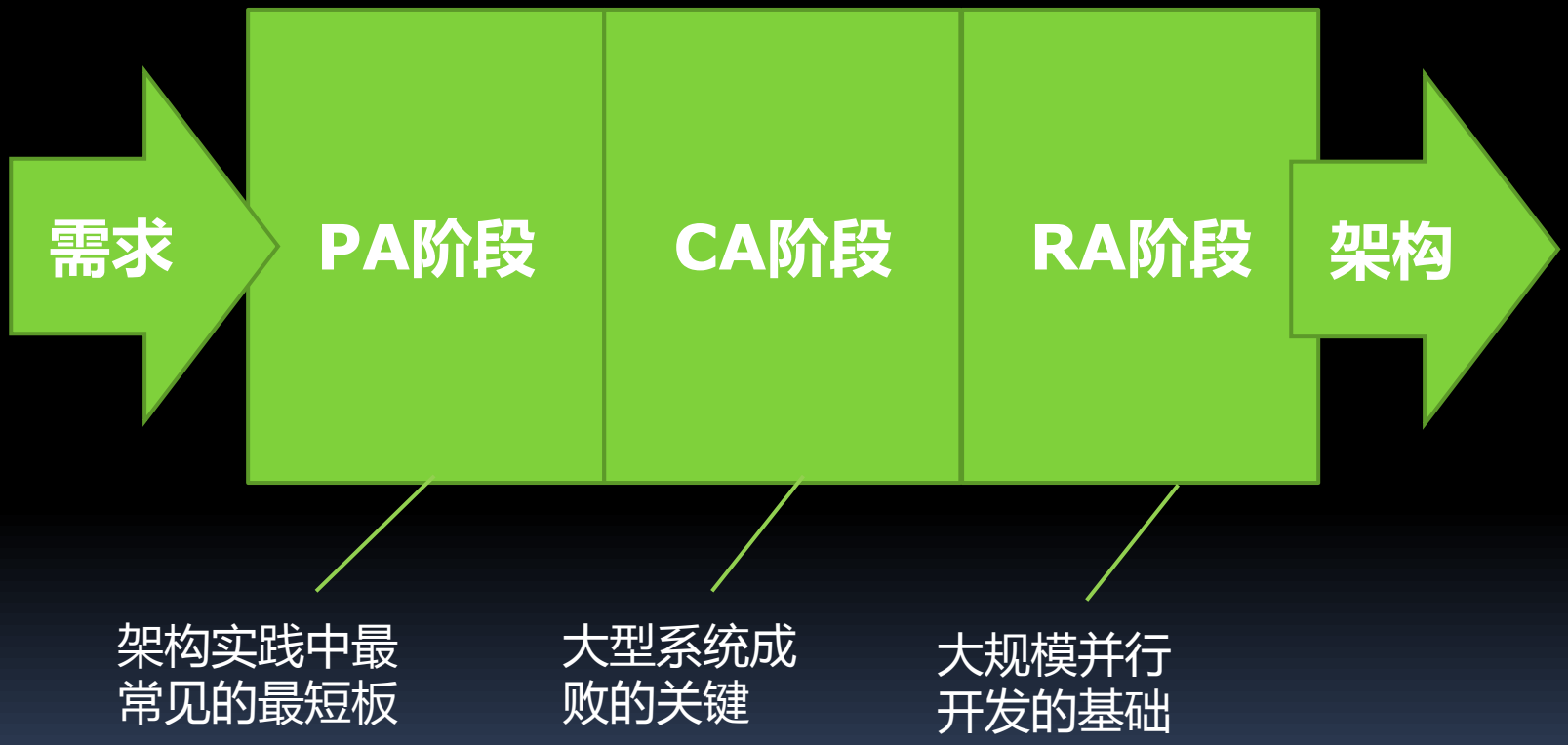
3.多阶段和多视图

- 架构设计不能一蹴而就，分阶段实施更现实。
- 3阶段
  - 阶段一：把握需求特点，确定架构驱动力。
  - 阶段二：根据重大需求，确定概念架构。
  - 阶段三：细化架构设计，关注不同试图。
- 5视图
  - 在第三阶段，引入5中视图，分别对架构进行细化设计。

4.有效的实践方法

- 在架构设计过程中，提供了一些具体的可操作的做法，来帮助我们执行架构设计的具体工作。
  - 逻辑架构设计的一些原则。
  - 质疑驱动的逻辑架构设计的整体思路。
  - 初步设计的方法。
  - 需求矩阵。
  - 约束的几种处理方法。
  - 非功能需求的思考等。

方法体系：3个阶段，1个贯穿环节



PA阶段：Pre-architecture

- 阶段目标：全面理解需求，从而把握需求特点，进而确定架构设计的驱动力。
- 最大误区：架构师是技术人员不必懂需求。
- 工具：需求矩阵。

CA阶段re：Conceptual-architect

- 阶段目标：考虑包括功能、质量、约束在内的所有方面的需求，构造核心架构并作高层分割。
- 最大误区：概念架构 = 理想架构。
- 工具：UML图、目标-场景-决策 表。

RA阶段：Refine-architecture

- 阶段目标：在概念架构基础上，对其进行细化，并从5个视角对架构进行细化设计。
- 最大误区：架构 = 模块 + 接口。
- 工具：UML图、目标-场景-决策 表。

贯穿环节：持续关注非功能需求

- 非功能需求不是架构的核心，但却关乎架构的成败。
- 利用 目标-场景-决策 表分析和设计非功能需求。

End