软件体系结构概论

Gaurav5582

14 Dec 2015

摘要

这是一个软件架构的介绍。

1 什么是软件架构，为什么我们需要它?

像任何其他复杂的架构一样，软件必须建立在坚实的基础上。不考虑关键场景，就不能针对共同的问题做出设计，或不能理解关键决策，其长期后果就会使你的应用处于危险之中。导致在软件中添加新特性会使得可维护性差和生产率低。

软件架构是一个技术蓝图，说明如何将系统构造成子系统（模块），同时优化诸如性能、安全性和可管理性等共同质量属性。

Philippe Kruchten、Grady Booch、Kurt Bittner 和 Rich Reitman 从 Mary Shaw 和 David Garlan 的作品（Shaw 和 GARLAN 1996）的基础上导出并提炼了一个建筑学的定义。他们的定义是：

> 软件架构包括一组关于软件系统组织的重大决策，包括系统组成的架构元素及其接口的选择；这些元素之间的协作所指定的行为；将这些架构和行为元素组合成更大的子系统；以及指导该组织的架构风格。软件架构还包括功能性、可用性、弹性、性能、重用性、可理解性、经济和技术约束、权衡和美学关注。

在《企业应用体系结构模式》一文中，Martin Fowler 在解释架构时概述了一些常见的重复主题。他把这些主题确定为：

> 系统的最高级别分解成其部分；难以改变的决策；系统中有多个架构；架构重要的东西可以在系统的一生中发生变化；最后，架构归结为重要的东西。

在《实践中的软件体系结构（第二版）》一文中，Bass，Clements 和 Kazman 定义架构如下：

> 程序或计算系统的软件架构是系统的架构或架构，包括软件元素、这些元素的外部可见属性以及它们之间的关系。架构与接口的公共方面有关；只需与内部实现无关的元素细节的私有细节不是架构。

架构的目标是识别影响应用程序架构的需求。良好的架构降低了与构建技术解决方案相关的业务风险。一个好的设计是足够灵活的，能够处理在硬件和软件技术以及用户场景和需求中随时间而发生的自然漂移。

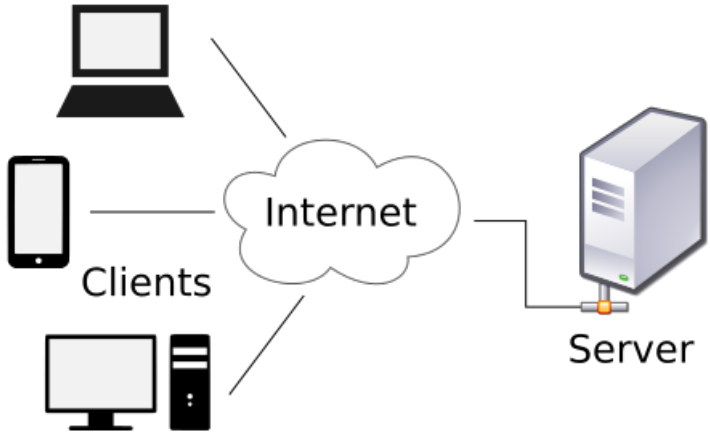
软件架构提供了一些相当相关的好处，包括：

- 更高的生产率。对现有软件添加新的特性更容易，因为架构已经到位，并且每个新代码的位置都是事先知道的。
- 更好的代码维修性。维护现有软件更容易，因为代码的架构是可见的和已知的，因此更容易发现错误和异常。
- 更高的适配性。新的技术特征，如前端不同，或添加业务规则引擎更容易实现，因为你的软件架构创建了一个明确的关注点分离。
- 炒作不可知论。最后但并非最不重要的一点，它将允许你看到行业中的炒作和时尚——或者说有很多——根据你目前的架构，如果需要的话，你就可以适应这些炒作和时尚。

2 软件架构模式与风格

2.1 客户/服务器架构

Client/Server 是一个软件架构模型，由两部分组成，即客户端系统和服务器系统，它们通过计算机网络或在同一台计算机上通信。客户机-服务器应用程序是由客户机和服务器软件组成的分布式系统。客户机-服务器应用程序提供了更好的分担工作负载的方法。客户端进程总是启动到服务器的连接，而服务器进程总是等待来自任何客户端的请求。

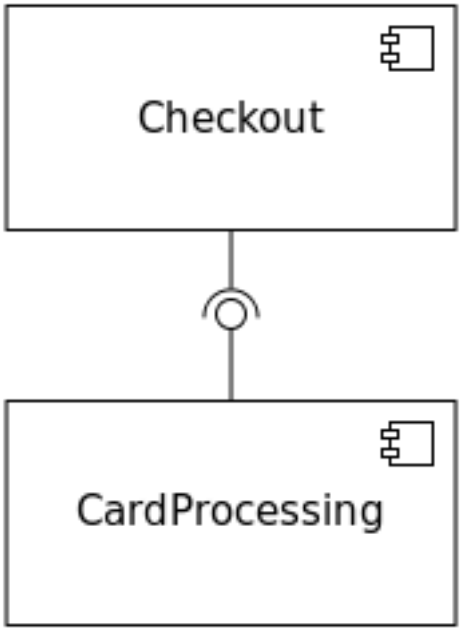


2.2 基于构件的架构

基于组件的架构着重于将设计分解成单独的功能或逻辑组件，这些组件表示包含方法、事件和属性的良好定义的通信接口。它提供了更高级别的抽象，并将问题划分为子问题，每个问题都与组件分区相关。

基于组件的架构的主要目标是确保组件的可重用性。组件将软件元素的功能和行为封装到可重用和自部署二进制单元中。

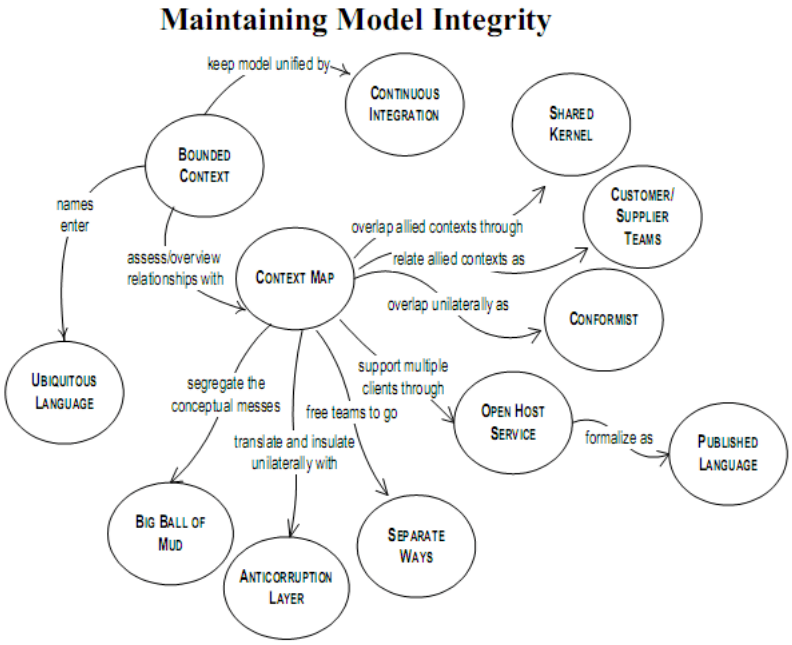
在 UML 2 中表示的两个组件的示例。负责促进客户订单的结账组件需要卡处理组件对客户的信用卡/借记卡（后者提供的功能）进行收费。



2.3 领域驱动设计

领域驱动设计（DDD）是一种构建符合核心业务目标的高质量软件的方法。它强调领域专家、开发人员、UX 设计者和其他学科之间的协作，以创建反映业务需求的领域模型。这涉及到对通用

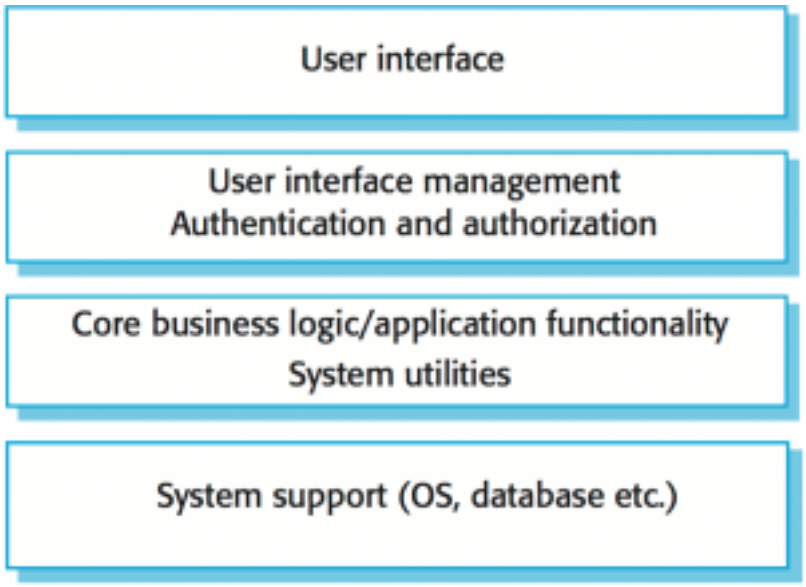
术语（即无处不在的语言）的认同，识别企业实体、他们的行为和关系，并将它们组织成一种清洁和模块化实现的方式。



战略领域驱动设计中的模式及其关系

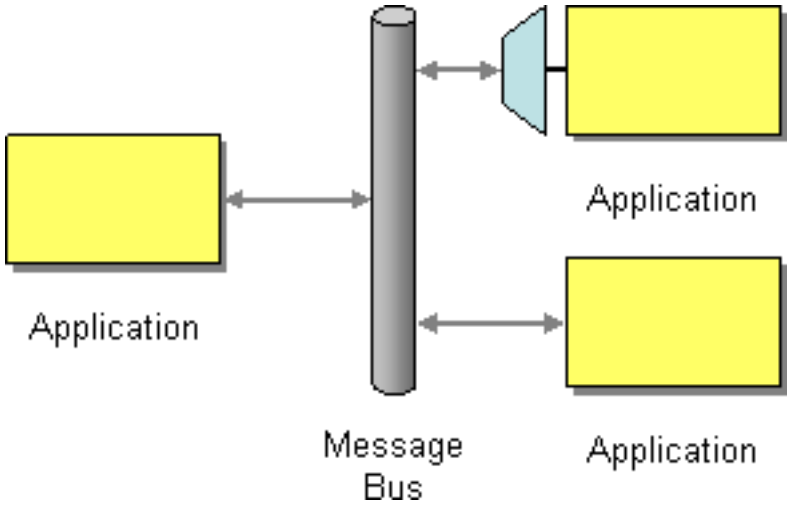
2.4 分层架构

分层架构侧重于将应用程序内的相关功能分组成不同的层，这些层在彼此的顶部垂直堆叠。每个层中的功能是由共同的角色或责任相关的。层之间的通信是明确的和松散耦合的。适当地分层应用程序有助于支持关注的强烈分离，进而支持灵活性和可维护性。



2.5 消息总线架构

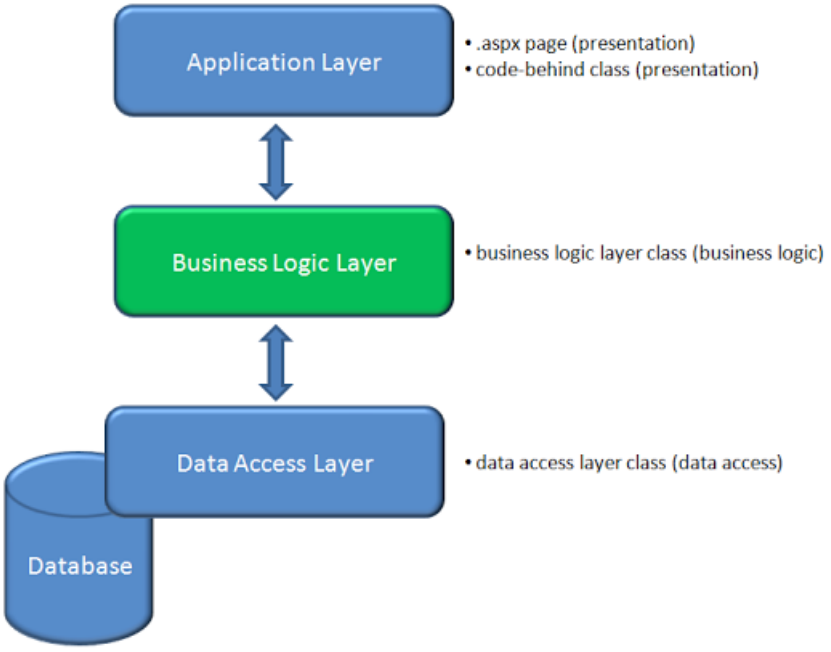
消息总线架构描述了使用软件系统的原理，该软件系统可以使用一个或多个通信信道接收和发送消息，以便应用程序能够交互而不需要知道彼此的具体细节。它是一种设计应用程序的样式，其中应用程序之间的交互是通过在公共总线上传递消息（通常是异步的）来实现的。最常见的消息总线架构的实现使用消息传送路由器或发布/订阅模式，并且通常使用消息队列（如消息队列）来实现。



消息总线

2.6 N 层/三层架构

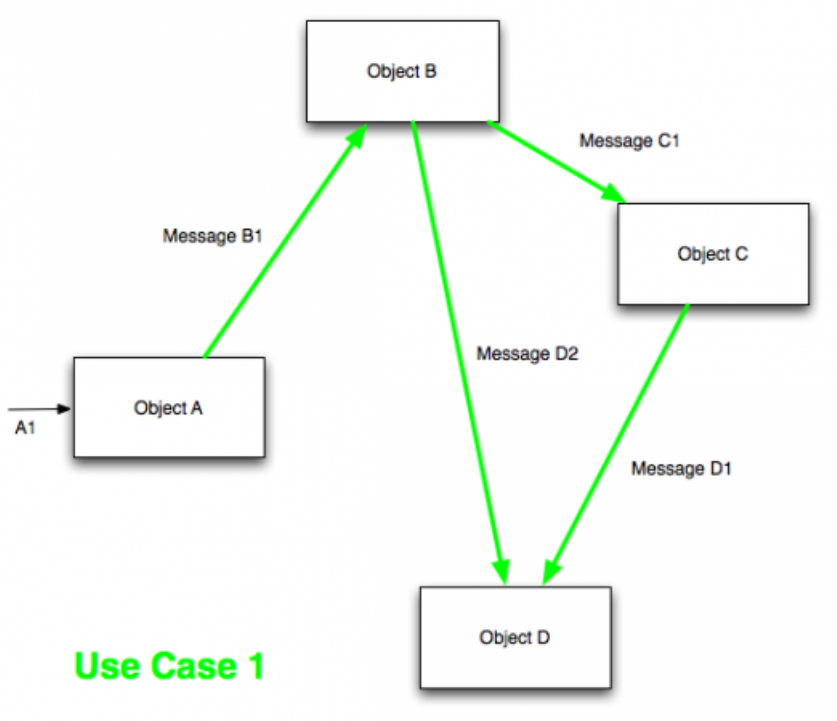
N 层/三层架构描述功能与分段样式大致相同，但每一段都是位于物理分离计算机上的层。它们通过面向组件的方法进化，通常使用特定于平台的通信方法，而不是基于消息的方法。



N 层/ 3 层架构

2.7 面向对象架构

面向对象架构是一种基于应用程序或系统的职责划分为独立的可重用和自给自足的对象的设计范式，每个对象包含数据和与对象相关的行为。面向对象的设计将系统视为一系列协作对象，而不是一组例程或过程指令。对象是离散的、独立的、松散耦合的；它们通过接口、通过调用方法或访问其他对象中的属性以及通过发送和接收消息进行通信。

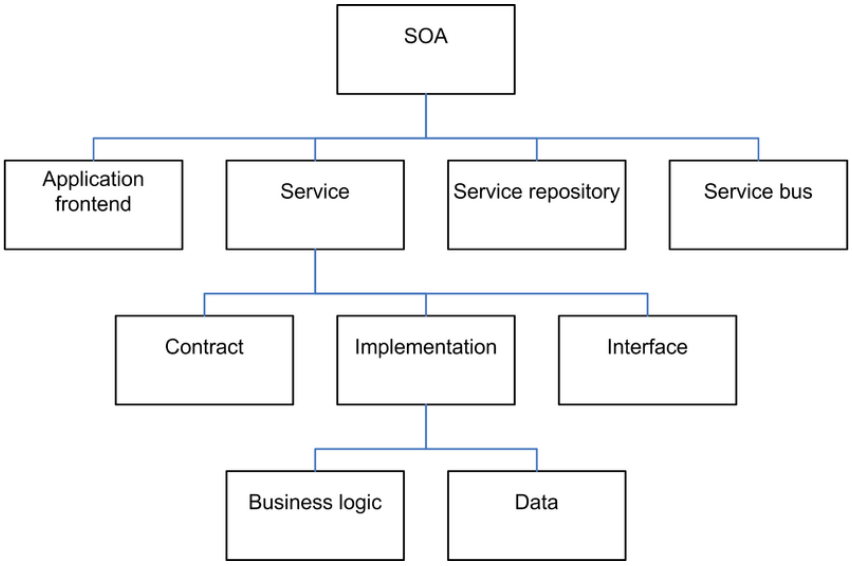


面向对象架构

2.8 面向服务的架构（SOA）

面向服务的架构（SOA）是一种用于创建基于服务的使用的架构的方法。服务（如 RESTful Web services）执行一些小功能，例如生成数据、验证客户或提供简单的分析服务。

SOA 架构的一个关键是，相互作用发生在松散耦合的服务中，这些服务是独立运行的。SOA 架构允许服务重用，使得在升级和其他修改时不需要从头开始。



面向服务的架构

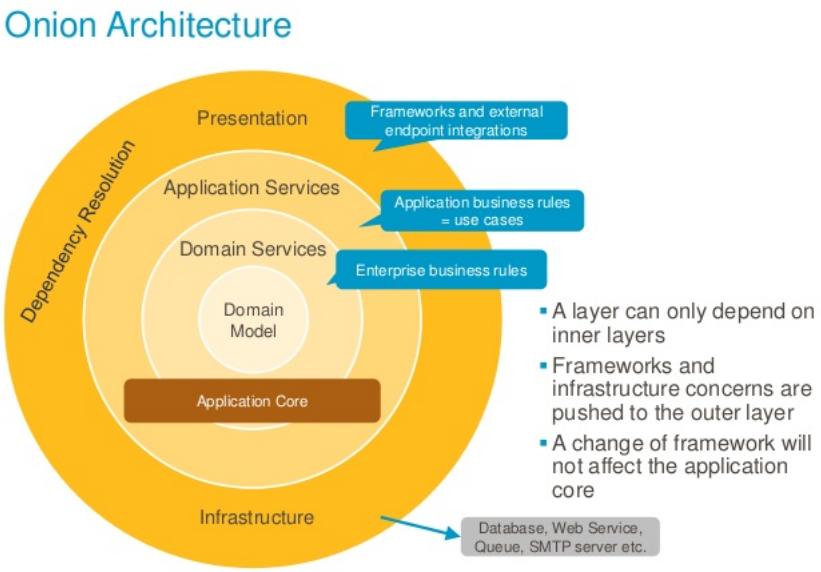
2.9 洋葱架构

* 洋葱架构 * 由Jeffrey Palermo提出。它类似于 Alistair Cockburn 提出的 * 六边形架构 *（端口和适配器）。

该方法的基本动机是避免通常与 N 层架构方法相关的层间依赖性。这是通过将所有基础设施（包括数据库）放置在问题域之外实现的。

将此原理应用于 N 层分层架构意味着反转依赖性流，使业务逻辑层独立于接口和基础设施。

这一原理导致了一个模型，其中业务逻辑处于架构的中心，附加的层围绕着同心环，如洋葱。



洋葱架构

3 综上所述

这是在一系列文章中的第一个，目标是涵盖与软件架构相关的广泛主题。
我们将在这篇文章中发表更多文章，敬请关注。你可以免费订阅这个博客，每周更新新文章。

4 References

- https://msdn.microsoft.com/en-us/library/ee658098.aspx
- https://msdn.microsoft.com/en-us/library/ee658117.aspx
- https://en.wikipedia.org/wiki/Software_architecture
- http://sanderhoogendoorn.com/blog/index.php/why-do-we-need-software-architecture
- https://simple.wikipedia.org/wiki/Client-server
- http://www.tutorialspoint.com/software_architecture_design/component_based_architecture.htm
- https://archfirst.org/domain-driven-design
- http://www.cs.ccsu.edu/~stan/classes/CS530/Notes14/06-ArchitecturalDesign.html
- http://searchsoa.techtarget.com/definition/service-oriented-architecture
- https://blog.apnic.net/2015/08/18/architecture-at-apnic-the-onion-architecture
- http://www.slideshare.net/kindblad/application-architecture-32607528