

**《软件架构实践》读书笔记**



**专 业 软件工程**

**姓 名 沈蕾薇**

**学 号 2022141461132**

**指导老师 毌攀良**

**成绩分数**

2025年 1月 2 日

**《软件架构实践》读书笔记**

**姓名 沈蕾薇 学号 2022141461132**

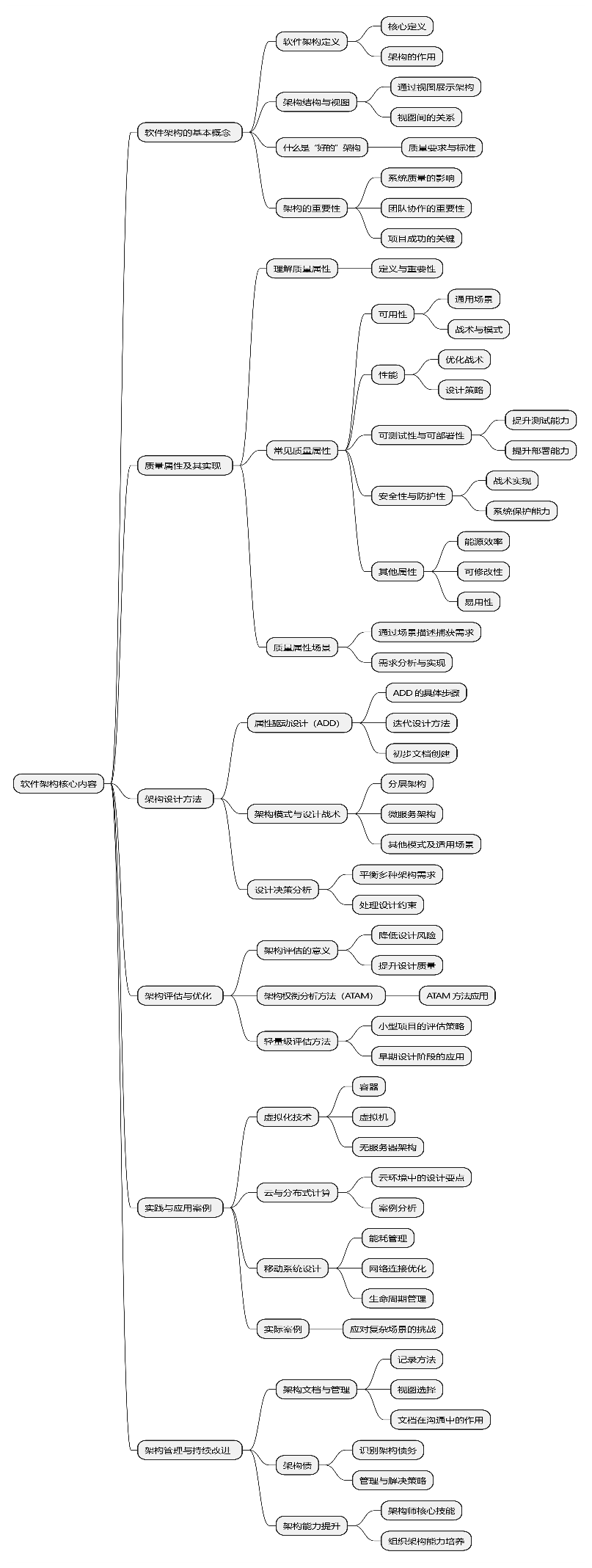
1. **引言**

在软件开发过程中，架构设计是决定项目成败的关键因素之一。一个优秀的软件架构不仅能提高系统的性能、可扩展性、安全性等质量属性，还能显著降低开发成本和风险。然而，随着系统复杂性的增加，设计决策变得越来越具有挑战性。在学习软件开发的过程中，我知道在项目中经常会面临诸如性能瓶颈、模块耦合度过高、系统难以扩展等问题。然而，我对架构的理解仅停留在一些基础概念上，不足以应对复杂项目中的实际需求。因此，我希望系统地学习软件架构的理论和实践方法，深化理解软件架构的核心思想，并探索如何将理论与实践相结合，从而提高软件系统的整体质量和性能。

《软件架构实践》（第4版）由软件架构领域的权威人士伦·巴斯（Len Bass）、保罗·克莱门茨（Paul Clements）和瑞克·凯兹曼（Rick Kazman）共同撰写，是软件架构领域的经典之作。书中全面探讨了软件架构的核心概念、设计方法、评估技术以及实际应用案例，为读者提供了全面的架构知识体系。它不仅涵盖了分层架构、微服务架构等常见模式，还详细介绍了属性驱动设计（ADD）和架构权衡分析方法（ATAM）等实用工具。同时，同时，书中也涉及云服务、虚拟化技术等最新技术趋势，是一本内容丰富、结构清晰的宝贵参考书。对软件架构师和开发者而言，这本书具有很高的实用价值。

1. **核心内容梳理**

在读完《软件架构实践》后，我结合书本目录和自己的理解，将书本核心内容分为了6个部分。下面是关于书中核心内容梳理的思维导图以及各部分的总结。



**2.1软件架构的基本概念**

软件架构是系统设计的核心，它定义了系统的基本结构和行为，明确了系统组件之间的关系以及如何协同工作。架构不仅仅是代码的组织形式，更是一种高层次的设计，提供了一个可以指导开发、测试和维护的框架。

软件架构的结构通过不同的视图展现，每个视图从特定的角度反映系统的设计，比如逻辑视图展示系统的功能性需求，开发视图描述模块划分，而物理视图则强调部署和运行环境的布局。通过这种多视图的方式，架构师可以更全面地分析系统，确保设计能够覆盖不同利益相关者的需求。

好的架构通常以明确的质量目标为核心，确保系统在可扩展性、性能和安全性等方面达标。它能最大限度地减少项目中的复杂性和不确定性，为后续开发奠定坚实的基础。此外，软件架构的合理设计对团队协作也至关重要，它明确了开发职责的边界，减少了因模块接口不清晰而导致的沟通问题。

**2.2质量属性及其实现**

软件系统的质量属性是评估架构设计优劣的关键。不同的系统对质量属性的要求各异，因此理解和实现质量属性是架构师的重要职责。例如，可用性是指系统在给定时间内的正常运行能力，这通常通过冗余设计、故障恢复机制和运行监控等战术来实现。另一个常见的质量属性是性能，它与系统的响应时间、吞吐量等指标直接相关，架构师需要通过减少不必要的通信、优化算法以及合理分配资源等方式进行性能优化。

除了可用性和性能，书中还详细讲述了可测试性和可部署性。通过模块化设计、清晰的接口规范以及自动化工具的引入，可以极大地提升系统的测试和部署效率。对于一些关键系统，安全性更是至关重要，通过设计上的访问控制、数据加密等方式，系统能够更有效地抵御外部攻击。其他如能源效率和易用性等特定属性，在一些应用场景中也有着重要地位。为确保这些属性能够满足需求，架构师常通过质量属性场景的方式对需求进行捕捉和分析。

**2.3架构设计方法**

在架构设计阶段，属性驱动设计（Attribute-Driven Design，ADD）是一种常用的系统化方法。ADD 将质量属性作为设计的核心驱动力，以属性目标为出发点指导模块划分、接口定义以及组件选择。这种设计方法强调了逐步迭代的过程，从最核心的质量目标(ASR)出发，逐层分解问题，最终形成初步的架构设计文档。

在实际设计中，架构模式和设计战术的应用能为架构师提供更多的参考。例如，分层架构常用于企业应用，微服务架构则适用于需要灵活扩展和部署的系统场景。架构设计并非单向决策，而是需要在质量、成本、开发难度等多方面进行权衡。合理的设计决策通常需要参考现有的经验和实践，同时考虑到项目的独特性。

**2.4架构评估与优化**

架构评估是保证设计质量的重要手段。通过定期评估，架构师可以识别潜在的设计缺陷并加以优化。架构权衡分析方法（ATAM）是一种经典的评估工具，它通过分析不同设计决策对质量属性的影响，帮助架构师找到最佳的设计方案。ATAM 强调利益相关者的参与，确保评估结果能够全面反映实际需求。

此外，对于小型项目或设计初期，可以采用更加轻量级的评估方法，这些方法灵活性较强，适合快速迭代和验证。

**2.5实践与应用案例**

在实际应用中，软件架构的设计往往需要结合具体的技术背景和业务需求。以虚拟化技术为例，容器化和无服务器架构的兴起为现代系统带来了更高的灵活性和资源利用率。在云计算领域，分布式系统的架构设计强调了数据一致性、弹性伸缩和故障容错能力。对于移动应用而言，设计还需要充分考虑设备的能耗、网络连接的不稳定性以及生命周期的管理策略。这些领域的实际案例提供了大量宝贵的经验，帮助架构师应对复杂场景下的设计挑战。

**2.6架构管理与持续改进**

软件架构的管理贯穿于系统开发的全生命周期。架构文档是管理的重要工具，它记录了架构设计的核心思想和实现细节，为开发团队提供了清晰的指导。架构文档的质量直接影响到团队的沟通效率和开发效率，因此需要在视图选择、文档更新和版本控制等方面做出规范化要求。

在实际开发中，架构债是一个普遍存在的问题。如果在早期设计中未能充分考虑长期需求或妥协于短期目标，可能会为系统带来难以解决的架构问题。通过定期审查和优化，团队可以及时发现并解决架构债务。与此同时，提升架构师的能力以及组织的架构管理水平也是确保项目成功的重要因素。

1. **个人感悟**

**3.1理论部分**

在阅读学习这本书的过程中，我对软件架构的设计和管理有了系统的认识。在之前，我只知道可用性、安全性、微服务架构等名词，但并不清楚它们在设计过程中发挥着着什么样的作用，同软件架构有什么样的关联。在阅读完这本书后，我能把之前零散的知识点串联起来了。

在这些内容中，我对质量属性和属性驱动设计（ADD）方法两部分有着深刻的印象，感觉收获颇丰。

质量属性是软件架构的核心目标，它决定了系统的非功能性表现和用户的整体体验。通过阅读，我深刻认识到质量属性不仅仅是单一的技术需求，它更是对系统的全方位考量。各质量属性之间相互影响，一种质量属性的实现必然会影响其他质量属性，这种影响有积极的，也有消极的。

书中还将质量属性通过场景描述和设计战术系统化地分解，使得抽象的质量要求变得可以操作。例如，在分析可用性时，书中建议通过“减少故障发生频率”和“提升故障恢复速度”这两个战术来设计解决方案。这种战术化思维让我意识到，架构设计并不是笼统的需求满足，而是通过具体技术与场景的结合，逐步实现目标。

ADD 是我在本书中印象最深刻的设计方法之一，它将软件架构设计流程分为清晰的步骤，从需求到结构逐步实现。ADD方法分为五个步骤：①选择要设计的系统元素②确定所选元素的ASR③为所选元素生成设计方案④清点剩余需求，为下一次迭代选择输入内容⑤重复步骤1-4，直至完成。

相比传统的自顶向下或自底向上的设计方法，ADD 的最大优势在于其以质量属性为驱动，通过明确关键的架构需求来引导设计决策。此外，ADD 还让我意识到要重视架构设计中的利益相关者沟通。在复杂系统中，不同利益相关者（如客户、开发者、项目经理）对质量属性的关注点往往不一致，而 ADD 的框架促使我们在设计初期就与各方充分沟通，减少了后期因需求不明确而导致的返工。

**3.2实践部分**

这本书书名叫做《软件架构实践》，通过对本书的学习，我理解了质量属性与 ADD 方法的理论基础，于是我尝试将其应用于实际项目中，根据学到的内容设计了一个仓库管理系统。我认为理论与实践的结合能够帮助更好地解决实际问题，并验证设计方法的可行性和有效性。下面是关于仓库管理系统的架构设计：

首先，根据仓库管理系统的业务需求，我总结了该系统的ASRs。仓库管理系统的ASRs包括可扩展性（系统需要支持动态扩展，以适应新增商品种类或处理高峰期的用户访问）、性能（确保高并发条件下，系统能够快速响应库存查询和更新操作）、安全性（通过严格的权限控制和数据加密保护敏感数据）、可用性（保证系统在硬件故障或维护期间的服务连续性）等。

接着基于上述质量属性，我为每个质量属性构建了多个场景来细化需求，下面是有关可扩展性的两个场景示例：

（1）可扩展性场景1：添加新仓库

来源: 系统管理员

刺激: 系统管理员在后台管理界面请求增加一个新仓库，并配置相关的商品种类和库存数量。

响应: 系统成功创建并配置新仓库，支持新仓库的数据同步，且新仓库信息可以无缝集成到现有库存管理系统中。

响应度量: 新仓库配置和数据同步在30分钟内完成，新仓库的数据可以即时与其他仓库数据同步，且系统无卡顿或崩溃。

环境: 系统负载较轻，当前没有出现资源瓶颈或冲突。

系统：库存管理系统，特别是仓库管理模块及跨仓库数据同步模块。

（2）可扩展性场景2：服务模块扩展

来源: 研发团队

刺激: 研发团队需要向现有系统中添加一个新的业务模块，如自动化库存补货模块。

响应：新模块无缝集成到现有系统中，且不影响现有库存管理功能的正常运行，同时能够与其他业务模块（如销售和采购模块）进行数据交互。

响应度量: 新模块在48小时内成功部署并与现有系统集成，且系统保持高可用性（99.9%）并无性能下降。

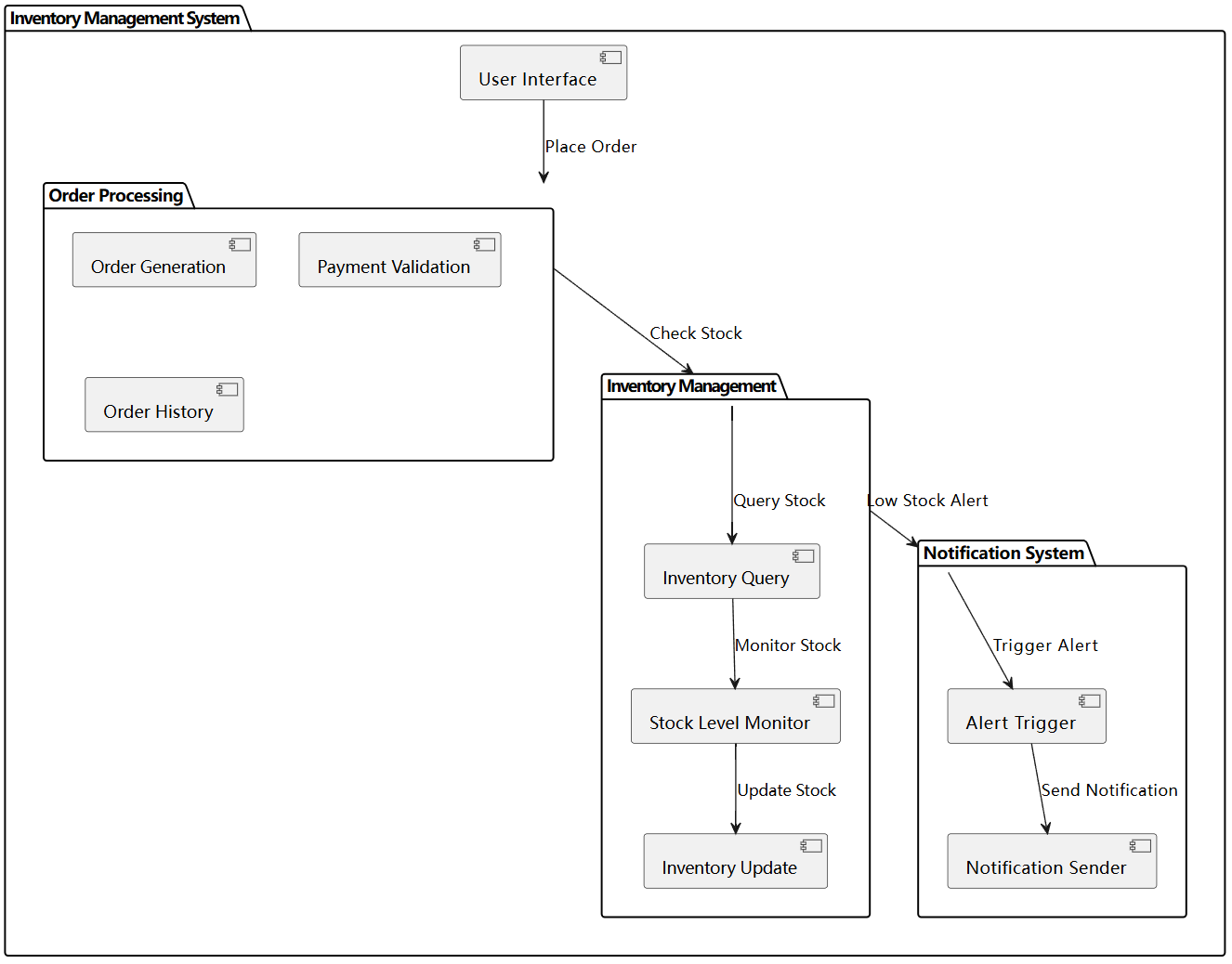
环境: 系统部署在高负载环境下，正处理大量实时数据和事务请求。

系统: 库存管理系统，特别是微服务架构和业务模块扩展部分。

为满足上述质量属性，我基于ADD方法设计了系统的模块分解试图、并发视图和部署视图。这些视图分别展示了系统的静态结构、动态行为以及硬件部署方案。

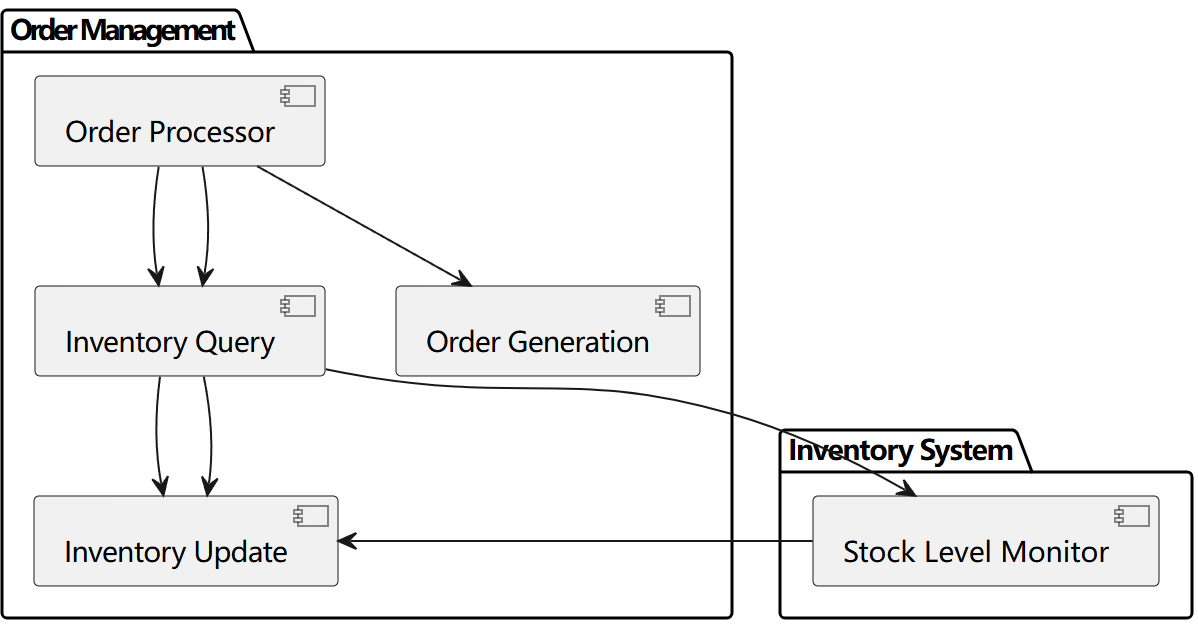
首先是模块分解视图。它提供容器来承载系统职责并展示模块间的交互方式，帮助设计可修改的系统架构。通过模块分解，系统的功能可以被清晰地划分到不同的模块中，从而降低耦合性，提高系统的可维护性。

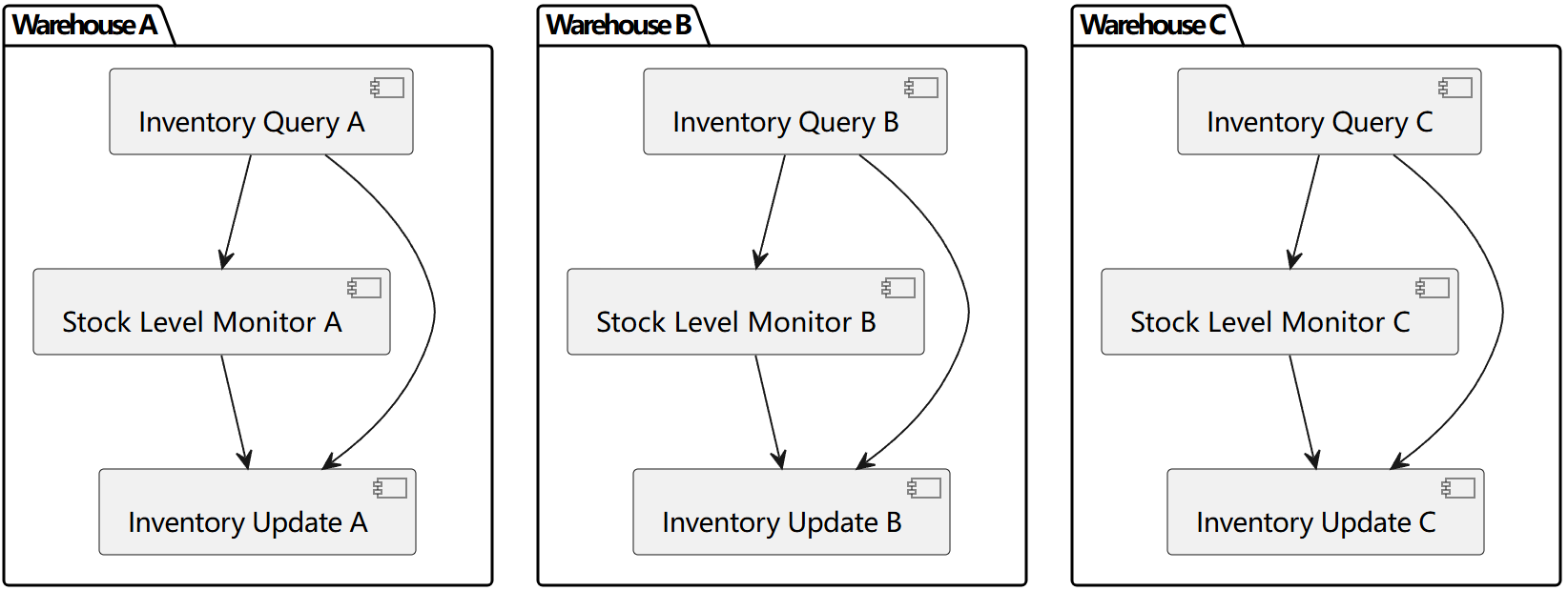
该系统的分解视图展示了库存管理系统的功能模块设计结构。整个系统被分为四个主要模块：用户界面、订单处理、库存管理和通知系统。系统通过模块化设计实现功能分离，提高系统的可维护性和扩展性。



然后是并发视图，它用于建模系统的动态行为，例如并行活动、任务同步和系统交互。其主要目标是优化系统的性能

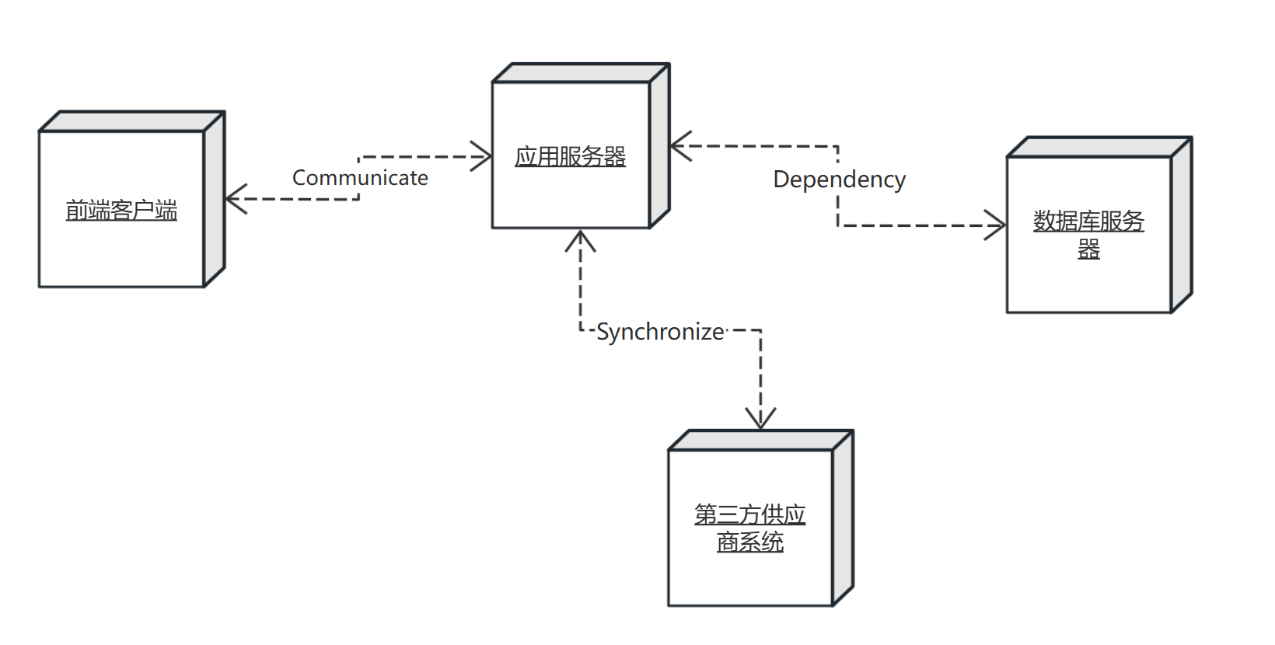
在本系统中，并发视图针对订单处理、库存更新和跨仓库数据同步等并行任务，设计了多个并发组件（如订单处理器、库存查询模块等），以提升系统性能。下面两个并发视图分别是订单处理与库存更新的并发视图和多仓库管理的并发视图





最后是部署视图，用于设计系统在硬件上的部署方式，特别是在存在多个处理器或专用硬件时，帮助分析硬件部署带来的附加职责。其目标是支持满足系统的质量属性需求，例如性能和可靠性。

对于本系统，系统通过双向通信接口与外部供应商系统进行数据交互，实现库存信息的实时更新与同步，同时保证了模块间的解耦性。



以上是我运用ADD 方法设计仓库管理系统的实践过程，可能这个设计过程并不完整甚至有错误，但通过这次实践，我对软件设计的相关内容有了更深刻的认识，对书中所讲的架构设计理论有了更加立体的理解：质量属性驱动的架构设计是从需求到实现的桥梁，而系统视图则是展示设计决策的最佳工具。

我切身感受到了以质量属性为目标，可以明确设计决策的方向，同时量化的场景描述也为后续测试和优化提供了清晰依据。模块分解视图、并发视图和部署视图帮助我梳理了系统职责与动态行为，确保架构设计清晰又灵活。

1. **总结**

通过阅读《软件架构实践》（第4版），我系统地学习了软件架构设计的核心理论与实践方法。这本书从基本概念入手，逐步深入到质量属性的实现、架构设计方法、评估技术以及实践应用案例，帮助我全面理解了软件架构的复杂性和重要性。

这本书不仅是一本技术指南，更是一种思维方式的培养。它加深了我对软件架构设计的理解，激发了我进一步探索和学习的兴趣。通过学习，我更加体会到软件架构设计是一门平衡艺术，它需要在性能、扩展性和灵活性之间进行取舍，最终达成整体系统的最优设计。这种系统化的思考方法，将成为我未来工作和学习的重要指导。

未来，我希望能将书中学到的知识更多地应用到实际项目中，通过实践巩固理论，进一步提升自己的软件开发能力与架构设计水平。我相信，扎实的软件架构设计能力不仅能帮助我设计出更加高效、可靠、灵活的系统，还能让我在未来更加从容应对复杂的技术挑战。