《软件项目管理》

个人读书笔记

（2024-秋）

|  |  |
| --- | --- |
| 书名： | **恰如其分的软件架构-风险驱动的设计方法** |

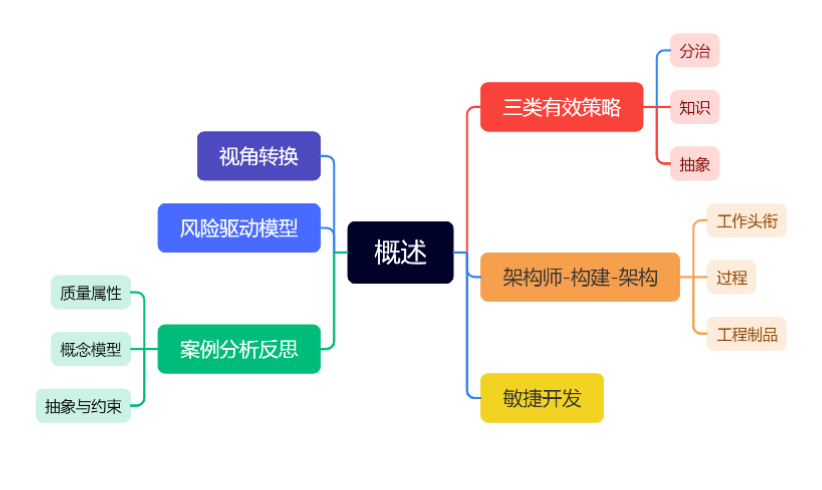
|  |  |
| --- | --- |
| 姓名： | 黄家俊 |
| 学号： | 2021141510022 |

**第一章 概述**

1.章节回顾

这一章首先从介绍分治、知识与抽象三个策略开始，阐述了这些策略如何帮助开发者应对软件复杂度和规模的增长。接着，结合Rackspace公司的三个不同版本的系统架构案例，分析了不同架构在质量属性上的差异，如可修改性、可伸缩性和延迟时间，并强调了概念模型的重要性，接着又讨论了从GOTO语句旧编程范式到新架构抽象的接受过程。然后，作者通过类比通俗易懂地阐述架构师角色、构建过程和工程制品(架构)三者的概念及关系，同时还简单介绍了风险驱动模型。最后围绕敏捷开发者的架构，作者强调了“恰如其分”和概念模型对于架构的重要意义。

2.知识梳理



3.体会感悟

本章主要探讨了软件架构的核心概念和实践价值。通过案例分析，我认识到在构建系统时，应该选择最符合当前需求的架构，并在此框架下实现功能。架构设计不仅仅是技术问题，更是权衡和决策问题，随着技术的发展，我们需要不断更新我们的思维模式和工具集。分治、知识与抽象这三个策略能帮我们在实际开发中将复杂问题分解为易管理的小部分，让开发更加高效。此外，风险驱动的架构设计为我提供了一种新的视角，即从风险的角度出发，合理分配设计资源。总之，架构师不仅要关注技术细节，还要考虑风险、团队协作以及项目的长期发展。

4.重要观点/语句摘录

* “架构是软件系统的宏观设计。”
* “软件架构正是这样的武器。它能帮助解决软件系统的复杂度及规模增长带来的问题。”
* “失败的风险有多高，你付出的努力就有多大。”
* 谚语云: “只见树木，不见森林。”我们却需要从各种设计细节(树木)中去发现架构(森林)。
* “新的架构抽象理念并未否定过去，而是兼容并包。”
* “敏捷软件开发是对重型开发过程的一次反击。它强调有效地构建客户真正需要的产品。”

**第二章 软件架构**

1.章节回顾

这一章首先通过引用卡耐基梅隆大学SEI的定义来介绍了软件架构的概念，并用美国的国家架构的例子来解释架构对于理解系统中的帮助。接着，借助建造摩天大楼的类比讨论了高层架构的决策意图如何影响底层的细节实现，并以此为切入点围绕系统的功能、质量、约束等角度阐述了架构的重要性，分析架构在不同案例中发挥的重要作用。在简单介绍了架构族后，以汽车为例子引入了忽视、接受和提升三种处理架构的方式，以此阐述其相对应的设计方式，即与架构无关的设计、关注架构的设计、架构提升，以具体的案例说明了三种设计的递进转变性。最后，作者给我们普及了大型组织中的架构，包括企业架构师和应用架构师的职责及面临的挑战。

2.知识梳理



3.体会感悟

阅读这一章后我对软件架构有了更全面的认识，架构的设计不仅关系到软件的质量和性能，还会直接影响到项目的成败。特别是在大型或复杂的系统中，架构的重要性尤为突出。我特别认同将架构比作系统的骨架这一比喻，它非常直观地说明了架构的强健与否会直接决定系统的稳定性和可扩展性。作为软件开发者我们需要正确理解系统目标，通过运用合适的架构，以满足系统需求、提高开发效率并有效降低风险。在大型的公司组织中，架构的角色和职责可能会分散在不同的团队和角色之间，这需要我们具备良好的沟通和协调能力，以确保整个组织的架构目标一致。

4.重要观点/语句摘录

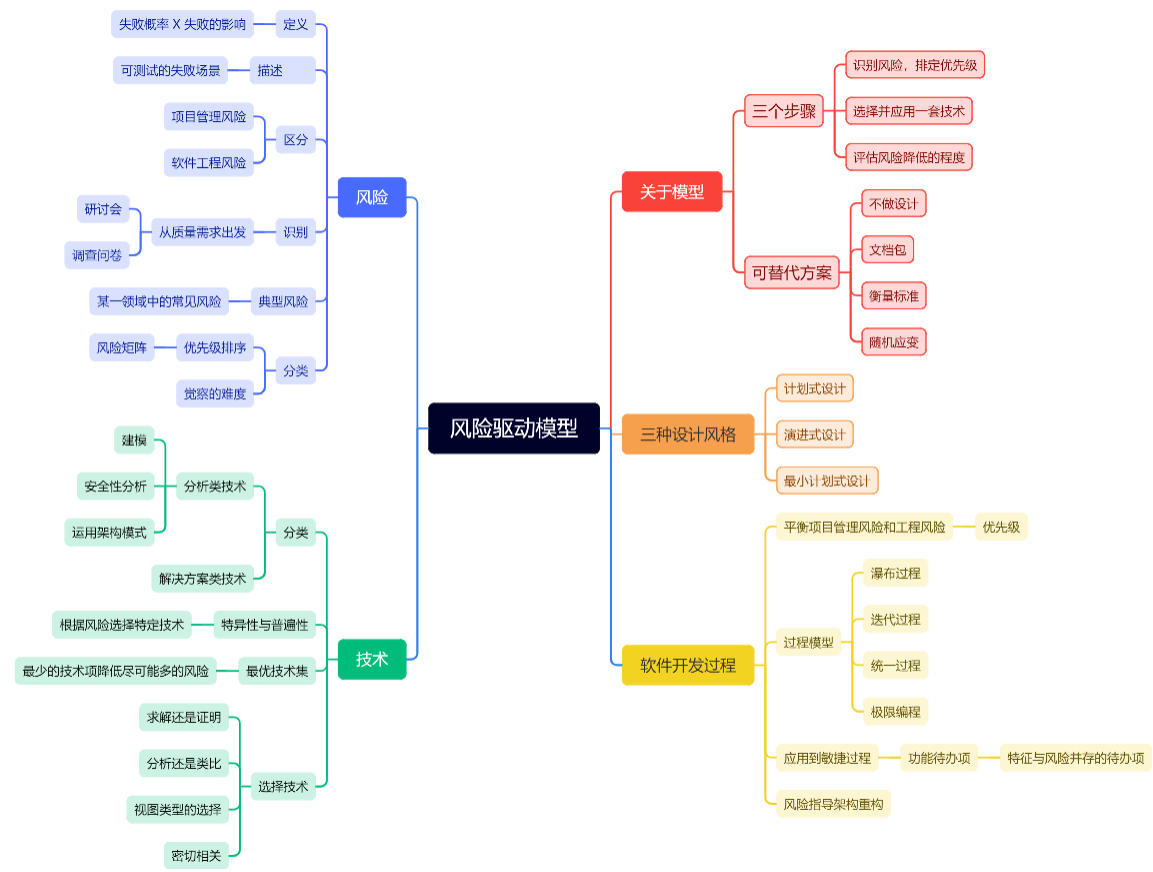
* “计算系统的软件架构是解释该系统所需的结构体的集合，其中包括：软件元素、元素之间的相互关系，以及二者各自的属性。”
* “架构是对系统恰如其分地施加约束，以便系统获得我们所需质量属性的一门艺术。”
* 骨架作为架构的隐喻,虽有不足却很有用。骨架为动物提供了整体结构,以支撑其行动。鸟之所以善飞,袋鼠之所以善跳,完全得益于它们各自的骨架。
* 尽管人和马的身体骨架都支持运输苹果到市场的功能,但在运输效率和数量上却相差甚远。选择一种架构使得系统能够工作并非难事,但在满足质量属性方面,有的选择是事半功倍,有的选择则会事倍功半。
* “任何系统都有其架构。采用架构无关的设计方式，也会产生一种架构，只不过此架构既没有经过审慎挑选，又并非开发者有意为之。”
* “架构无关的设计、专注架构的设计和提升架构的设计，这三种方式为开发者提供了不同的对待软件架构的策略。”

**第三章 风险驱动模型**

1.章节回顾

这一章首先从风险驱动模型的概念入手，介绍了如何识别和按优先级排序风险，并结合具体公司案例讲述了风险驱动模型的实际应用，引出对风险和技术的讨论。作者一方面介绍了关于风险的定义、识别和优先级排序，以及两类不同风险；另一方面又阐述了关于技术的分类、特性，以及在降低风险中的作用，并提供了技术选择依据。此外，在补充介绍了三种不同的设计风格后，作者围绕着软件开发过程与风险驱动模型的结合，简单介绍了瀑布、迭代、螺旋等不同的软件过程，并着重讨论了如何将风险驱动模型应用于敏捷过程。最后,通过回顾第一章的Rackspace公司的案例，说明了风险对架构重构的指导作用，并给出了可替代风险驱动模型的四种方案。

2.知识梳理



3.体会感悟

本章对风险驱动模型的介绍让我深刻体会到其在软件开发中的重要作用。它不仅仅是一个理论模型，更是一个实用的工具，帮助开发者在面对复杂多变的项目风险时，能够有的放矢地采取行动。书中提到了风险驱动模型可以帮助开发者根据风险的优先级和影响，合理分配资源和精力。这种平衡的方法让我认识到，哪怕在软件开发以外的领域中，我们也同样需要灵活地根据实际的情况调整工作和计划，而不是死板地遵循某种固定的模式。此外，章节中对于如何将风险驱动模型与敏捷过程相结合的讨论，也为我在敏捷开发实践中提供了新的视角和方法。总之，有效的风险管理是项目成功的关键，而风险驱动模型提供了一种实用的方法来实现这一目标。

4.重要观点/语句摘录

* “风险驱动模型可以指导开发者运用最小的架构技术集合去降低最紧迫的风险，以求事半功倍。”
* “只有将好钢用在刀刃上，才能构建成功的系统。这意味着只有当架构与设计技术受风险驱动时，才运用它们去消除风险。”
* “为追求效率，风险驱动模型遵循这一指导原则：为架构付出的努力应与失败的风险相称。”
* “风险驱动模型能够兼容并蓄，配合演进式设计、计划式设计与最小计划式设计。”
* “风险是工程师与项目经理之间的共享词汇。”
* “风险驱动模型遵循中庸之道，避免走极端，无论是编撰完整的架构文档，还是彻底放弃架构工作，都是不可取的。”
* “避免失败是所有工程之核心，可以运用架构技术去缓解风险。”

**第四章 实例：家庭媒体播放器**

1.章节回顾

本章通过家庭媒体播放器的开发案例，详细阐述了风险驱动模型在软件架构中的应用。首先，从团队沟通开始，介绍了新成员加入时如何通过阅读源代码和创建模块模型来防止出现架构偏移，同时说明其中的局限性。围绕媒体播放器的具体质量属性，讨论了如何通过识别和考虑质量属性的优先级来作出设计决策，进而引出运行时模型的概念，用于描述组件和连接器的行为。接着，作者讨论了COTS组件集成时可能遇到的问题，包括集成风险、可靠性、屏幕显示和延迟性，并展示了如何通过分析模型和原型设计来解决这些问题。最后，还探讨了元数据的一致性问题，介绍了如何通过领域建模来识别和解决潜在问题，并和其他模型进行了比较。

2.知识梳理



3.体会感悟

本章接着前一章进一步介绍了风险驱动模型。通过家庭媒体播放器的具体案例，让我更深刻地体会了风险驱动模型的运作方式，学会了如何通过创建模块模型、运行时模型和领域模型来降低沟通风险和技术风险。有效的沟通和准确的风险评估对于项目成功必不可少。在设计解决方案时，我们需要通过分析模型或者原型设计，来选择降低风险及检验风险是否降低的技术。此外，本章也初步解答了如何使架构建模最小化、如何运用软件架构技术、何时停止设计开始原型设计或编码，这些问题始终围绕着本书的主题，即恰如其分。这意味着，在实际开发过程中，要选择合适的技术来降低特定风险，在风险降低到可接受水平后适时停止架构活动，这些思想对于软件开发工作具有重要的指导意义。

4.重要观点/语句摘录

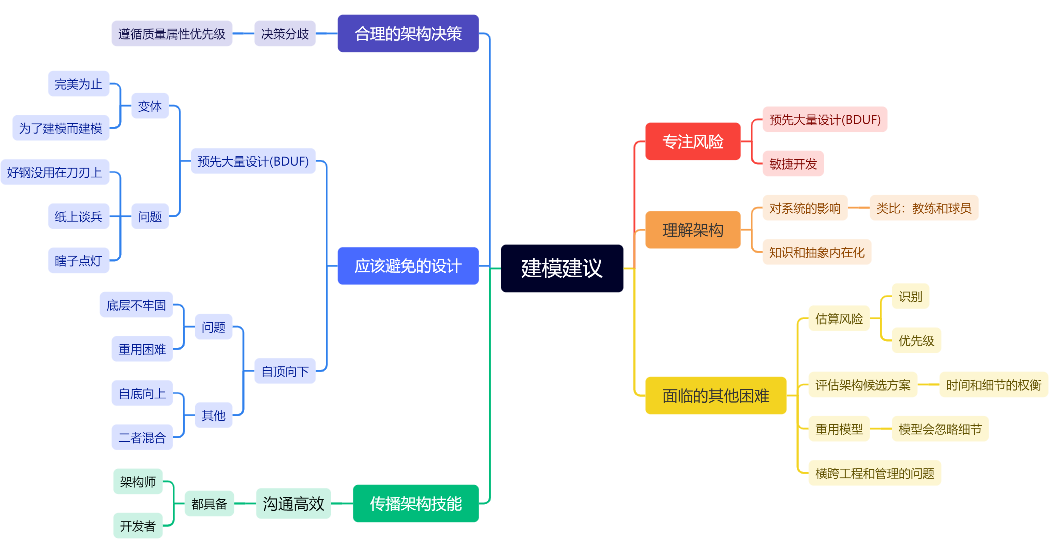
* “我们担心，在快速推动将原型转化为可运行的产品期间，新增的开发者会不经意地编写出违背我们设计的代码。这称为架构侵蚀或架构偏移。”
* “将代码作为唯一的沟通工具是有限制的。”
* “我们应定期对风险进行重新评估，一旦风险降低，就应该停止架构活动。建模能力再强，也不可能构建出最为完整的模型。”
* “不要寄希望于运用架构技术能够彻底消除风险，也不要在文档编撰完毕或项目取消之前，仍在运用架构技术。”
* “我们决定编写一个功能场景，用于描述它们如何共同工作。”
* “模块模型包括了一些隐含的领域术语和技术细节。”
* “一个模块模型明确识别了各个模块以及它们之间的依赖。”

**第五章 建模建议**

1.章节回顾

这一章先介绍了通过风险驱动方法设计软件架构的策略及其与螺旋模型的结合，接着用教练对体育比赛的理解来类比，说明了软件架构专家视角下对系统的理解，强调了理解架构的重要性，并指出为了提高沟通效率，除了架构师外开发者也应具备一定的架构知识。随后围绕如何作出合理的架构决策，提出了一个能确保决策与质量属性的优先级相对应的决策模式。章节还讨论了预先大量设计和自顶向下设计的各自潜在的问题，并提出了相应的替代方案。同时，作者也依次分析了软件架构在估算风险、评估架构候选方案、重用模型和跨越工程和管理等环节中的种种挑战，最后通过案例比较特性驱动开发和风险驱动模型的差异，强调了在设计中考虑风险的必要性。

2.知识梳理



3.体会感悟

本章围绕风险管理在软件架构设计中的核心地位，给出了相应风险指导下的建模建议，在以往建模分析中如果过于关注于功能实现，而忽视了潜在的风险，则可能导致项目失败。本章让我认识到，通过识别和降低风险，可以更有效地指导架构决策，确保项目的成功。同时，我也意识到了理解架构的重要性，这不仅仅是架构师的责任，而是每个团队成员都应该具备的能力。这种全员参与的架构意识，能够提高团队的沟通效率和决策质量。此外，合理的架构决策模式也为我在面对设计权衡时提供了清晰的指导，帮助我更好地权衡不同质量属性之间的关系，最终在特性实现和风险管理之间找到平衡。

4.重要观点/语句摘录

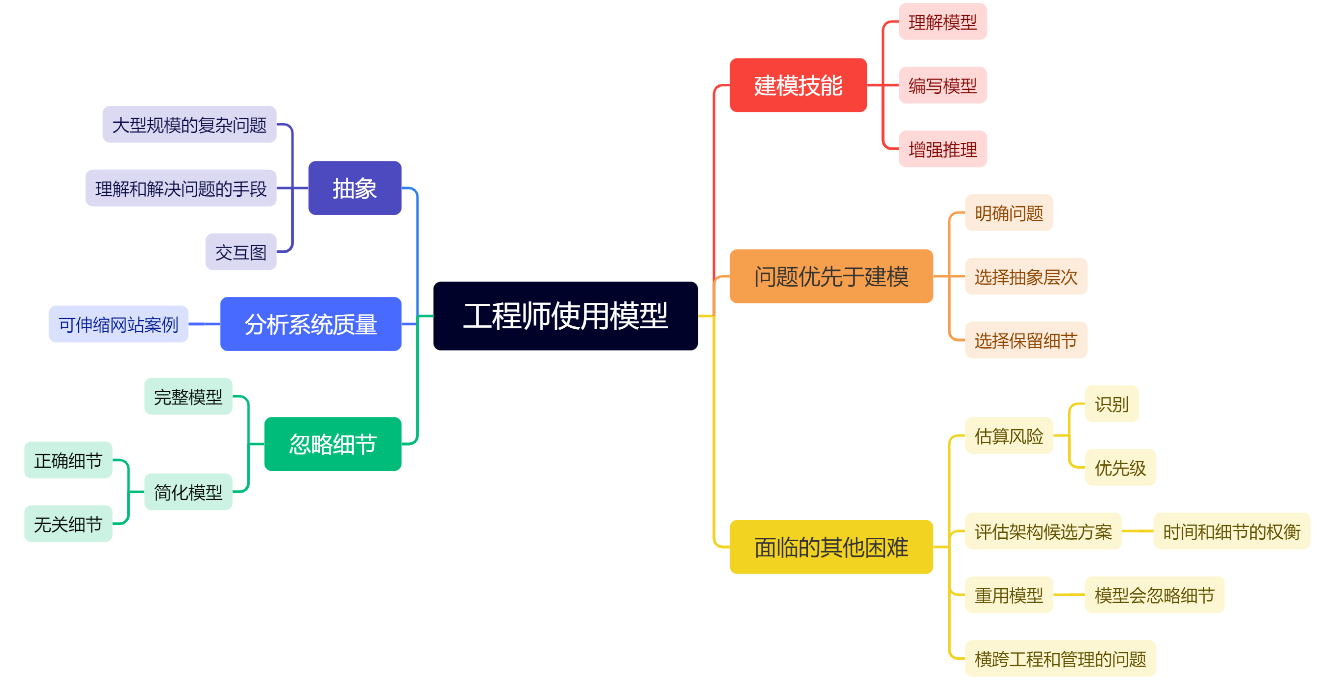
* “软件架构是对工程的关注，它与团队遵循的过程及团队成员的职位头衔无关。”
* “理解软件架构意味着你已经将建立的知识和抽象内在化了。”
* “设计关乎权衡，你不可能在质量属性方面做到面面俱到。架构决策必须是合理的，这意味着你做出的权衡需与质量属性的优先级对应。”
* “一个系统的架构能缓解工程风险，主要是针对质量属性的风险。”
* “风险可以被用来引导你进行适当的架构活动，帮助你决定何时停止建模并开始编码。”
* “很多人会建议你将注意力完全放在特性上。立足于这样的建议，我们已经看到有许多项目将时间浪费在特性与基础设施的实现上”

**第六章 工程师使用模型**

1.章节回顾

本章首先围绕着抽象这一概念，介绍了其重要性和作用。在面对大型或复杂问题时，直接分析变得不可行，因此需要利用更高的抽象层次来理解系统，同时借助一个数学问题的类比，说明了抽象模型如何帮助将问题的细节映射到模型中，再从模型中映射回现实世界的解决方案。接着，作者通过可伸缩性网站的案例，强调了建立模型在分析系统质量中的重要性，讨论了在模型中如何忽略无关细节保留正确细节，以有效地分析问题。此外，还以房屋建造设计为引子，探讨了建模的不同层次，从理解模型到编写模型，再到使用模型。最后，作者强调了在建立模型之前明确模型需要解决的问题的重要性，以确保模型的针对性和有效性。

2.知识梳理



3.体会感悟

在深入阅读第六章后，我更加深刻地理解了模型在解决工程问题中的核心作用。这一章让我认识到，无论是在软件架构还是其他工程领域，模型都是我们理解和处理复杂系统的重要工具。它不仅仅是一个简化的表示，而是一个能够揭示系统深层次特性和潜在问题的有力工具。书中“实质上，所有模型皆有谬误，然而部分模型却是有用的。”这句话对我有所启发，它提醒我，在构建模型时，我们不必追求完美无缺，而是要追求能够解决实际问题的有用模型。“提问在前，建模在后”不仅是一个简单的原则，也是一个关键的决策点，它决定了模型的方向和有效性。在日常生活中的决策过程，我们往往在没有明确目标的情况下就急于行动，导致结果往往不尽人意。

4.重要观点/语句摘录

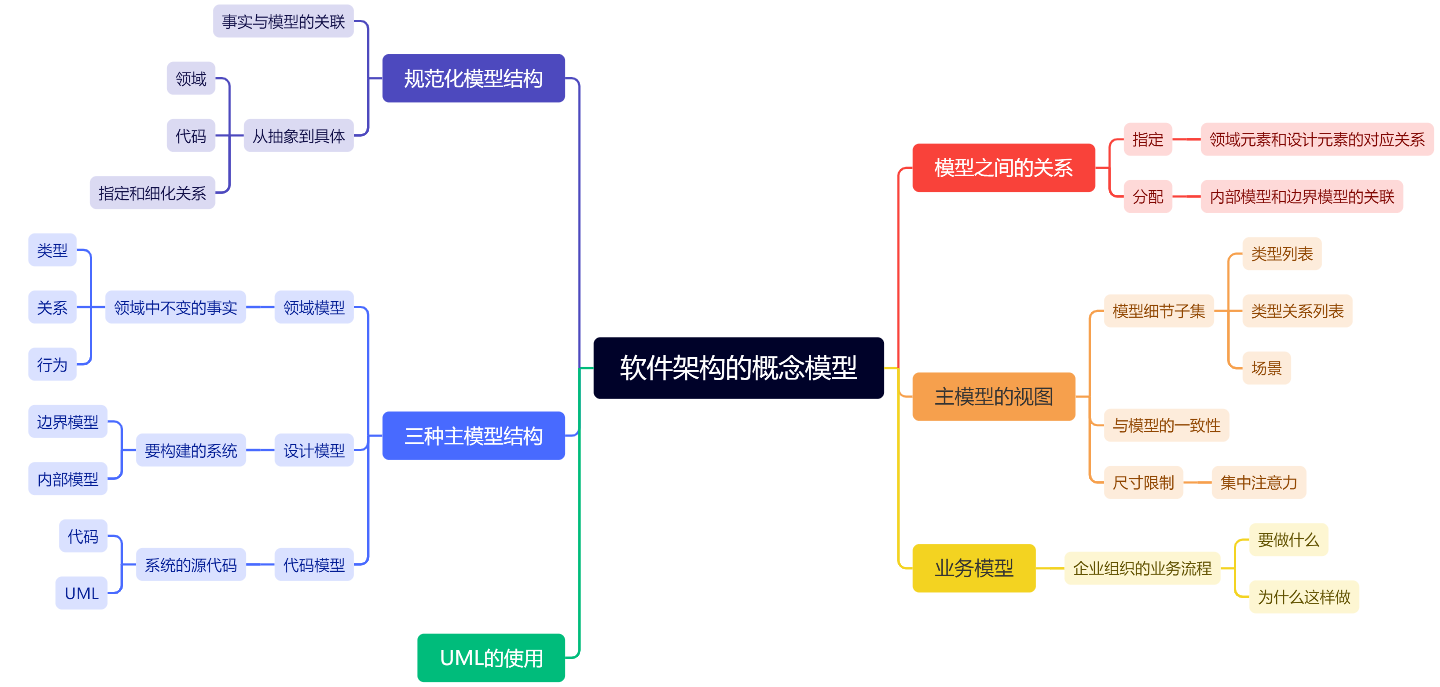
* “工程师的目标是将现实世界的问题转换为现实世界的解决方案。”
* “要了解系统，抽象会比直接查看源代码更为有效。”
* “提问在前，建模在后，即在建立模型之前，要明确自己需要模型解决哪些问题。如此才能更容易选择它的抽象层次，以及要包含的细节。”
* “一个模型能够帮助你组织系统的事实与细节。”
* “建立架构模型是一种理解和解决棘手问题的好方法，因为它们可以去除无关的细节，使得你能够将注意力放在主要部分及其相互关系上，作出预测，评估候选方案。”
* “模型各有所长。一个有助于预测响应时间的模型，可能无法帮助你发现安全漏洞。”
* “简单的问题可以直接解决，无须抽象。一旦面临复杂问题，工程师就需要将问题映射到一个抽象模型上，在模型内解决问题，再将解决方案转换为现实世界的方案。”

**第七章 软件架构的概念模型**

1.章节回顾

这一章首先从介绍规范化模型结构的概念入手，即用来关联所发现的事实与所建立的模型的一种组织形式，由此进一步详细说明了领域模型、设计模型和代码模型三种主要模型结构的区别和联系，并围绕不同模型之间的关系，说明了关系的指定与细化，其中指定关系确保不同模型中相似元素的对应关系，细化关系则关联相同事物的模型在不同抽象层次上的细节。接着，作者类比日常生活的一些例子，解释了主模型视图如何帮助我们筛选模型细节，以便集中注意力于相关因素，以及视图之间的一致性。此外也讨论了业务建模，解释了业务模型概念，以及与领域模型的区别。最后，还对比了不同专家提出的模型结构组织方式，并简单提及了UML的使用。

2.知识梳理



3.体会感悟

阅读本章我主要学习了在软件架构中的概念模型。通过规范化模型结构，我们可以更清晰地组织和理解软件的各个组成部分。领域模型、设计模型和代码模型的区分，让我明白了在不同阶段关注不同细节的重要性。指定与细化关系的概念对于我在设计软件时保持一致性和可追踪性至关重要，它提供了一种在不同抽象层次上关联和细化模型的方法。此外，视图的概念特别有启发性，它让我学习到在复杂的软件架构中，如何通过关注特定的细节来简化开发中的问题。总之，我不仅学习了概念模型的理论知识，这也为我提供了一种系统化的思考和解决问题的方法，帮助我在实际工作中更好地构建和管理软件架构。

4.重要观点/语句摘录

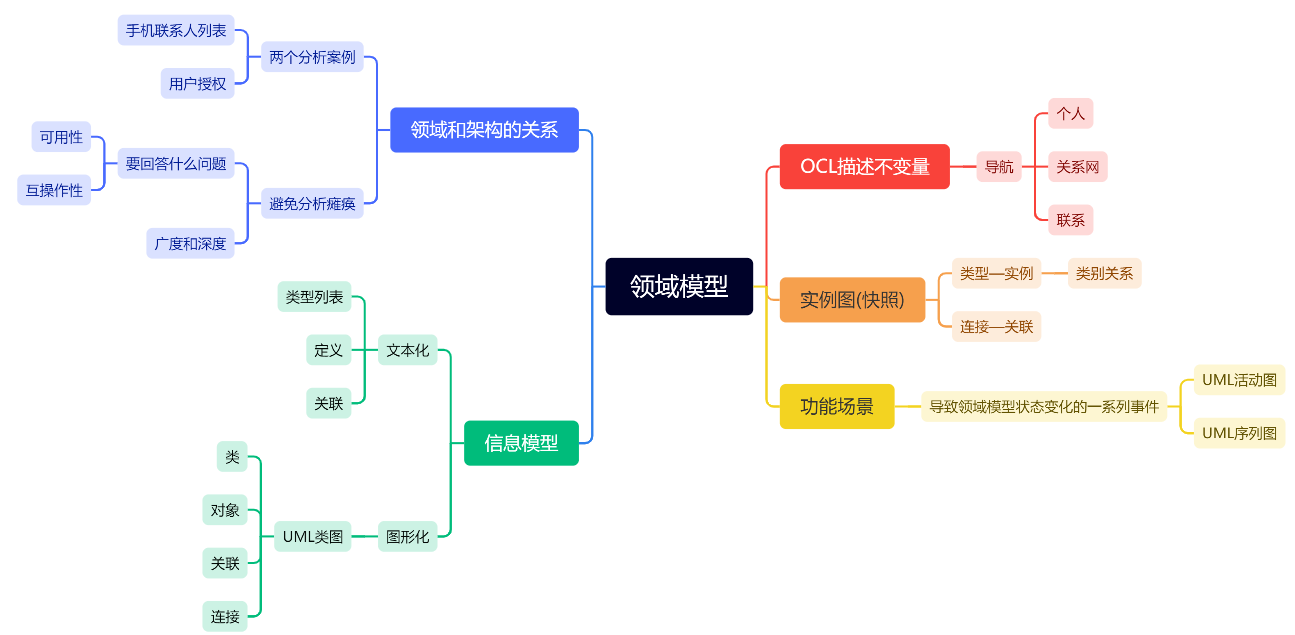
* “规范化模型结构的本质很简单: 它的模型范围从抽象到具体，使用的视图会深入每个模型的细节中”
* “教练的所见所得要多于新手，并不是因为他们的眼睛更犀利，而是因为他们掌握了概念模型，这个模型能帮助他们对观察到的内容进行分类。”
* “领域模型表达了与系统相关的现实世界的不变事实。”
* “代码模型既是系统的源代码实现，又相当于一个模型。”
* “细化是相同事物的模型在低层细节与高层细节之间的关系。”
* “视图允许我们仅选择模型细节中的一个子集。”
* “由于视图是一个单独主模型的所有投影,因而它们的细节都是一致的(或将是一致的)。为了分层地嵌套设计模型,需要使用细化来关联边界模型与内部模型。”

**第八章 领域模型**

1.章节回顾

这一章从领域与架构的关系开始，介绍了领域建模对于理解复杂领域和设计软件架构的重要性。通过两个案例展示了领域建模在实际应用中的价值。接着，章节深入探讨了信息模型，这是领域模型的基础，包括了类型定义列表和它们之间的关系，且通过UML图形化我们可以更清晰地理解这些关系。随后，围绕信息模型，作者进一步介绍了不变量，解释了如何在模型中表达约束和通过关联进行导航。此外，还介绍了实例图（即快照），它展示了领域模型中类型的具体实例，帮助我们理解模型的实例在某一时刻如何关联。最后，作者对功能场景进行了介绍，它描述了导致领域模型状态变化的一系列事件，并对如何借助UML通过一组场景来描述领域。

2.知识梳理



3.体会感悟

通过阅读第八章，我深刻意识到领域模型在软件开发中的重要作用。领域建模不仅仅是技术活动，更是一种沟通和理解的工具。它帮助开发人员和业务专家建立共同的语言，从而更准确地理解和解决问题。信息模型让我认识到了清晰定义领域中的类型和关系对于沟通和设计的重要性，而快照和功能场景则提供了一种直观的方法来展示领域模型的状态和变化此外，我也认识到了领域模型的简化性质，它需要我们做出选择，决定哪些内容是模型中必要的，哪些可以省略。这种选择的过程，实际上是对领域知识深度和广度的一种把握，也是对项目风险的一种评估。

4.重要观点/语句摘录

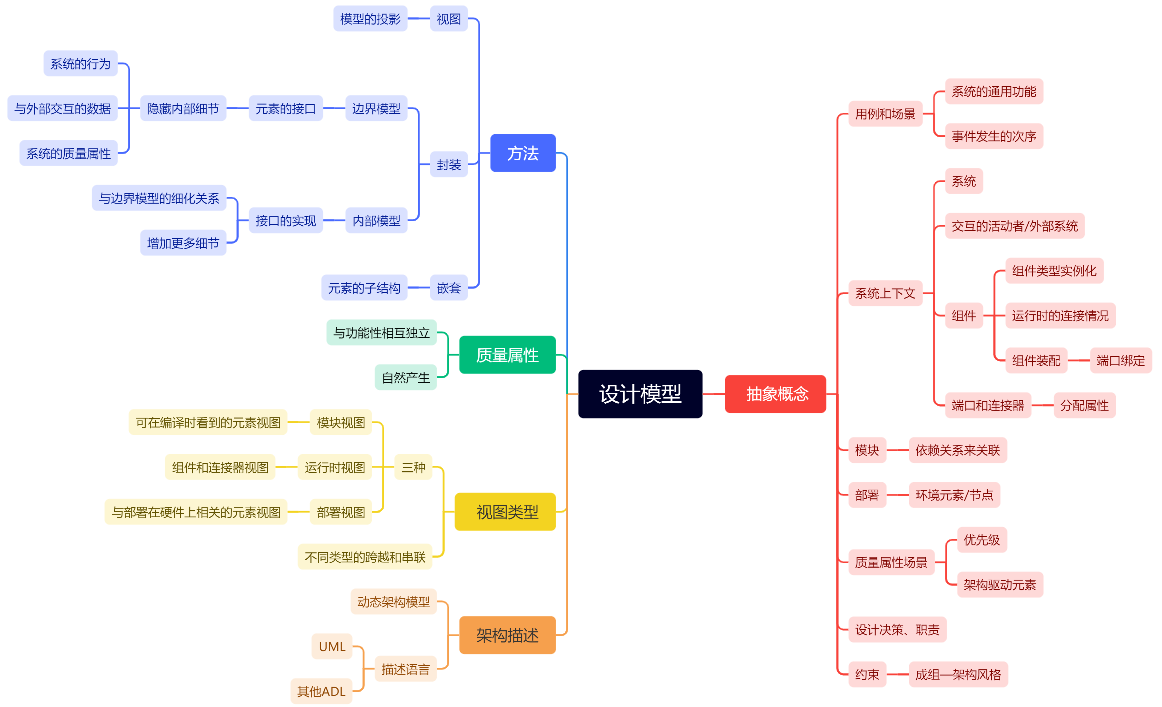
* “领域模型可以成为开发人员和主题专家之间建立共同语言的基础。”
* “信息模型也可以图形化绘制，与文本化版本相比较，图形化版本可以清晰地表达出类型之间的关系。”
* “功能场景通常简称为场景，表达了导致领域模型状态变换的一系列事件。”
* “将领域模型和设计模型分离是有好处的。如果没有分离，设计时就会碰到很多与设计无关的问题”
* “领域模型是对现实领域的一种简化，它必须决定哪些内容是包含在内的，哪些内容是排除在外的。”
* “领域模型不关注软件设计的细节，并且可以用一些简单的符号来表达，因此，它是一种和主题专家交流的有效方法。”

**第九章 设计模型**

1.章节回顾

这一章围绕设计模型的核心概念，介绍了设计模型作为包含所有设计细节的主模型，以及如何通过视图、封装和嵌套来简化和聚焦设计细节，由此进一步阐述边界模型和内部模型如何共同工作来描述系统的接口和实现细节，还简单讨论了如何在设计中权衡不同的质量属性。此外，通过Yinzer系统的设计案例，作者向我们展示了如何将设计模型的众多抽象概念应用于实际的系统设计中，包括用例图、系统上下文图、组件、端口、连接器等关键概念。同时，还介绍了三种不同的视图类型，探讨了不同视图之间的关系以及跨类型、不同类型的串联等问题。最后，通过简要介绍动态架构模型和架构描述语，进一步强调了架构设计的变化性和迭代性。

2.知识梳理



3.体会感悟

本章主要介绍了设计模型的相关知识体系，它不仅是连接领域知识和代码实现的桥梁，也是团队成员之间沟通的共同语言。书中提及的视图、封装和嵌套的概念，让我学习到了一种管理和简化复杂设计信息的结构化方法。同时，结合Yinzer系统的例子，我看到了如何将抽象的设计概念具体化，并在实际项目中应用的完整过程。此外，质量属性的讨论让我意识到，作为开发者，我们需要在不同的质量属性之间做出权衡，这需要深入理解业务需求和系统目标。最后，动态架构模型和架构描述语言的部分说明了架构设计是一个迭代的动态过程，需要不断地评估调整。

4.重要观点/语句摘录

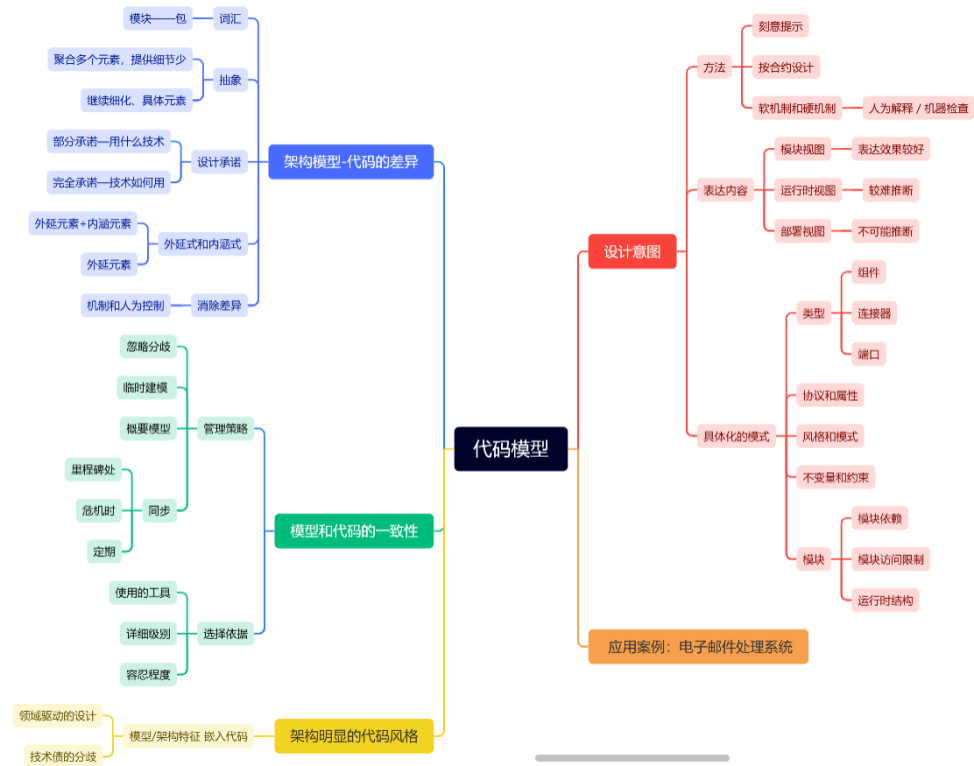
* “设计模型是包含所有设计细节的主模型。主模型的思想是一种方便实用的抽象，因为它解释了所绘制的那些图是如何相互联系在一起的。”
* “视图是模型的投影，我们可以使用视图，从而有选择地缩小对全面设计模型的关注范围。”
* “用例图擅长展示功能，而系统上下文图则擅长展示与系统交互的外部系统。”
* “架构风格限制了设计，从而使开发人员可以控制风险，并获得某种质量属性。”
* “动态架构模型描述架构在运行时是如何变化的。”
* “理想情况下，所有的系统都将最大化每一个质量属性，但实际的情况却是，必须设定质量的优先级。”
* “要分析系统，约束是必要的: 没有约束就意味着无法分析。没有约束，你的代码可以做任何事情”

**第十章 代码模型**

1.章节回顾

这一章首先从架构模型和源代码之间的差异性开始，介绍了二者在词汇表述、抽象、设计承诺和内涵外延等方面的不同。接着，阐述了如何管理和维护架构模型和源代码之间的一致性，提供了一系列管理策略以及不同策略的适用场景。然后，通过类比面向对象的编程风格，作者引入了架构明显的代码风格这一概念，强调了将架构特征嵌入代码中的重要性，并提出模型嵌入代码原理。接着，还探讨了在代码中应该表达哪些架构设计意图，并提供了一系列的模式来帮助开发者在代码中表达架构模型。最后，作者通过电子邮件处理系统的实例，展示了将本章所学的概念思想应用于实际的系统开发中。

2.知识梳理



3.体会感悟

这一章围绕架构模型和代码之间的关系展开论述，二者差异性除了技术层面，更在于表达设计意图时的不同。保持它们一致性对于软件架构的清晰性和可维护性至关重要。架构明显的编码风格为我提供了一种新的视角，让我意识到在代码中明确表达架构设计意图的重要性，这不仅有助于减少未来代码演化中可能出现的问题，也使得新加入的开发人员能够更快地上手项目。通过电子邮件处理系统的案例，我学习到了具体理论应该如何结合实际开发，作为一名软件开发者我在今后会更加注重架构模型与代码的一致性，同时养成在代码中表达设计意图的习惯。

4.重要观点/语句摘录

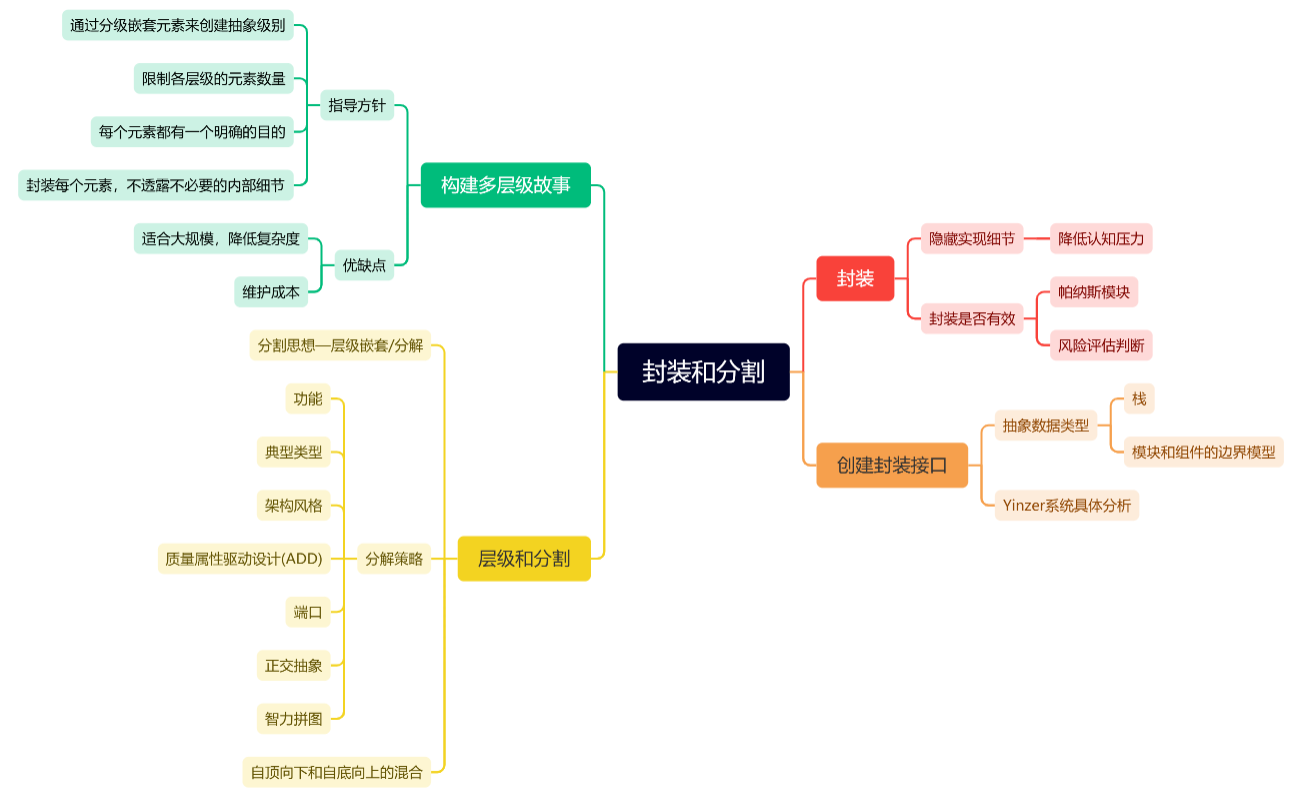
* “源代码既是最终的交付物，又是表达解决方案的媒介。”
* “架构模型不是最终的交付物，只有在和代码建立关联之后，它才是有用的。”
* “架构明显的编码风格，即把架构特征嵌入代码，从而降低丢失设计意图的可能性。”
* “在代码中保留设计意图，而不是保留在文档或图中，从而减轻文档压力，可以让新加入的开发人员快速上手。”
* “解决方案就像穿越迷宫的一条路：它将会成功地指引你从起点抵达终点，但不会告诉你为什么不选择其他的路。”
* “模块依赖是团队需要考虑的最常见的约束之一，编程语言难以表示这种依赖。”

**第十一章 封装和分割**

1.章节回顾

本章先通过新老机器比较的具体例子，说明了层级结构在认知上的优势，由此提出了构建多层级故事的必要性及指导方针，并进一步围绕分割这一核心思想，探讨了在不同系统中应用层级结构的挑战和限制，提供了相应的系统分解策略，包括功能、典型类型、架构风格、质量属性和属性驱动设计、端口、正交抽象和智力拼图等。接着，作者用面包机的类比，引入了封装的概念，并用亲身的开发经历阐述了封装有效性的判断，进而介绍帕纳斯模块。最后，在通过栈的例子进行代码分析以说明ADT的概念后，作者结合了Yinzer系统的具体案例详细描述了如何基于抽象数据类型的思想来创建组件接口。

2.知识梳理



3.体会感悟

在软件开发中，合理的分割和封装对于提高软件质量和可维护性的重要性。特别是在大型系统中，如何将复杂的功能和组件组织成易于理解的层级结构，对于开发人员来说是一个巨大的挑战。书中通过老机器和新机器的比喻，生动地说明了层级结构在认知上的优势，这让我意识到在设计系统时，追求的应该不仅是技术的实现，还有系统的可理解性和易用性。其中对帕纳斯模块的介绍让我了解到在设计模块时，应该隐藏可能变化的细节，以保持模块的稳定性和可替换性，这些策略对于我未来软件开发的实践有很大的帮助。

4.重要观点/语句摘录

* “如果这个层次结构构建得好，对每个看到它的人来说，就像在讲述一个故事，很容易理解。”
* “开发人员更喜欢组织良好的系统，而不是一个让他们头晕目眩，由类、模块及组件这些元素组成的海洋。”
* “ADD 的核心思想是，首先定义哪一种质量属性对于组件是最重要的，并通过质量属性场景表达出来。”
* “封装与分解密切相关。分解，认为问题要被拆分成更小的问题，封装则是说那些解决更小问题的方案应该被隐藏。如果你使用了一台烤面包机，可能并不关注加热零件是金属的，还是陶瓷的，只要转动旋钮，最后出来烤面包就可以了。”
* “有效的封装可以使用户理解 API，同时隐藏实现细节。”
* “封装最大的好处是降低了其他开发人员的认知压力。他们可以把组件或模块当做一个黑盒，只需要看这个黑盒的接口就可以了。”

**第十二章 模型元素**

1.章节回顾

本章从和部署相关的元素开始，介绍了节点和连接等环境元素，并围绕组件的概念，着重讨论了组件类型和实例、模块和组件的关系以及子组件的使用，同时澄清了一些关于它们的普遍误区。接着，结合图书馆系统的开发案例，作者解释了组件的装配，特别是组件细化的语义，再由组件间的通信引出了对连接器的介绍，比如连接器的属性和类型。随后作者继续使用图书馆的具体例子，先介绍了功能场景的构成和泛化方法，又说明了两种不变量的约束，再详细阐述模块、端口的概念。最后，作者还以质量属性为核心探讨了其特性、分类和权衡，以及质量属性QA场景，并介绍了关于系统元素的职责分配问题。

2.知识梳理



3.体会感悟

这一章介绍了众多的模型元素，每个元素都像一小块拼图，它们共同构成了软件架构的全貌。组件不只是代码的集合，它们在系统中具有实际的运行实体和结构，而连接器尽管它们在模型中可能只是简单的线条，但承载着重要的交互任务，是系统间通信的核心。此外，功能场景通过故事讲述的方式来描述系统行为的方法，让我对如何表达系统的动态特性有了更深的认识。文中对质量属性的讨论让我了解到这些非功能需求在系统设计中的重要性，以及如何在设计中权衡这些需求。在追求某些质量属性时，常常要在其他方面做出妥协，这往往需要考验架构师的智慧。

4.重要观点/语句摘录

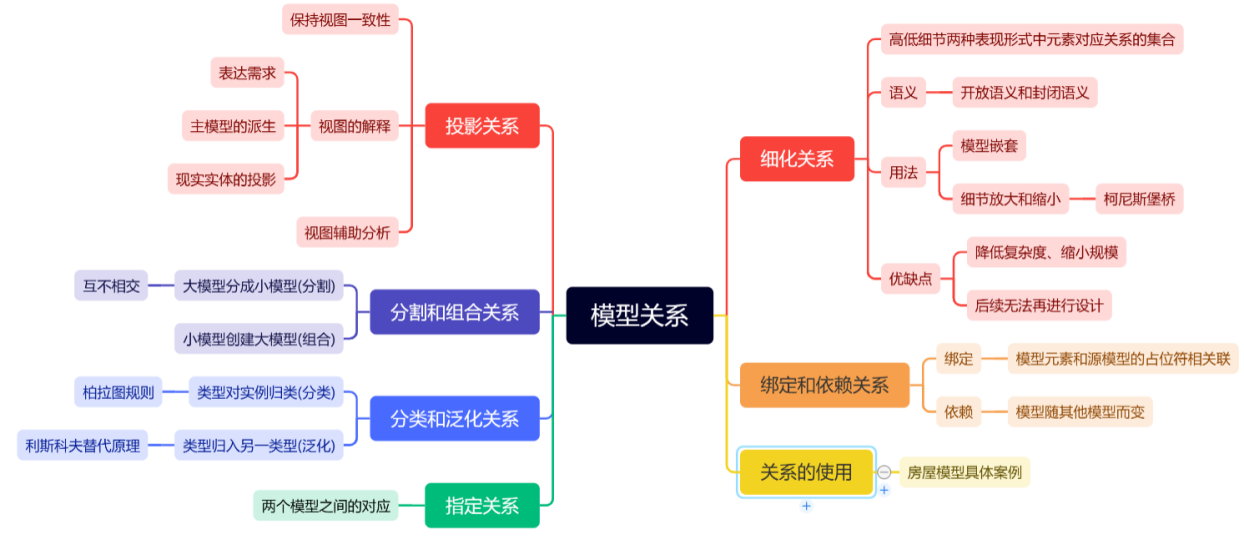
* “组件是软件架构中一个粗粒度的抽象。”
* “组件装配也称组件和连接器图，显示一个由组件、端口及连接器实例组成的装配图。”
* “连接器可以这样定义：‘两个或更多组件之间运行时的交互通道’。”
* “设计决策指导着前进的方向，同时约束着设计空间。”
* “功能场景讲述了一个关于那些元素如何随着时间而改变，以及相互之间如何交互的故事。”
* “质量属性是一种超功能需求，也称非功能需求或非功能属性。”
* “系统元素可以同时具有功能和质量属性方面的职责。”

**第十三章 模型关系**

1.章节回顾

本章介绍了投影或视图的概念，将其类比于地图投影讨论了视图间的一致性问题，进一步探讨了三种解释角度。接着，作者转向分割和组合、分类和泛化这些两两成对的关系，具体地阐述了这些关系如何帮助我们理解和分析，并补充介绍了柏拉图规则和利斯科夫替代原理。指定关系紧随其后，作者探讨了如何在不同域之间建立桥梁对应，特别是在现实世界和问题域模型之间。此外，作者还解释了细化关系，并以房屋图为例介绍了细化的开放和封闭语义。最后，在简要介绍了绑定与依赖关系后，继续沿用房屋的案例展示了如何在系统上下文中应用以上所有关系。

2.知识梳理



3.体会感悟

通过这一章的学习，我对模型关系在软件架构中的重要性及其多样性有了较为全面的认识。书中不仅系统地梳理了各种模型关系，如投影、分割、组合等，而且还通过房屋建筑的比喻，将抽象的概念具体化，更易于理解。投影视图不仅仅是展示模型的一个方面，也是沟通和分析工具，在多团队协作的项目中保持其一致性是一大挑战。此外，在实际的软件开发中我们经常需要在高层次的规划和低层次的实现之间穿梭，细化关系教会我如何将复杂的问题简化，使其更易于管理和解决。总之，理解这些关系有助于我们预测和控制变化，从而设计出更稳健和可维护的系统。

4.重要观点/语句摘录

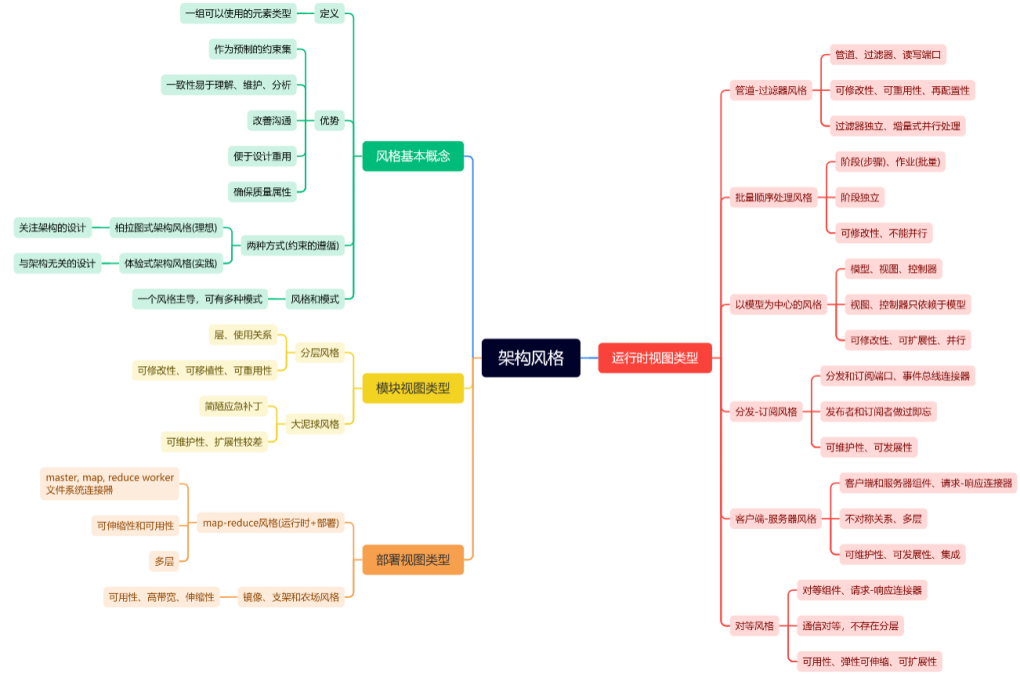
* “投影显示了一个模型细节的已定义的子集，显示的内容有可能经过了一些转换。投影可以抛弃细节，比方说，忽略了国家边界的地图。”
* “视图帮助你应对两个主要的问题：复杂性和规模。”
* “利斯科夫替代原理提供了一种对泛化的简单测试: 子类型一定可以用它的超类型替换。如果时尚房屋是简约房屋的子类型，那么你可以睡在时尚房屋里面，也可以睡在简约房屋里面。”
* “只有对庞大而复杂的问题做些转化，使之可以放进你的头脑，你才可能去构建更庞大、更复杂的软件系统。细化让你简化问题的复杂性、压缩问题的规模，从而使之变得易于处理。”
* “一个模型的改变会导致另一个模型发生变化，这就是依赖关系。”
* “设计师常常用二维视图来进行工作，他们常常用自己的大脑来保证各个视图的一致性。”

**第十四章 架构风格**

1.章节回顾

这一章先引入架构风格的基本概念，介绍了其作为预制约束集如何提高设计的一致性和可理解性以及如何改善沟通和促进重用。接着，讲解了柏拉图式风格与体验式风格的差异，并讨论了以架构为中心的设计方法以及架构模式与风格之间的区别。在简略列出架构风格的目录，作者按照视图的类型，详细地介绍了包括分层风格、大泥球风格、管道-过滤器风格等等十种不同的架构风格，还结合具体的代码与开发案例，进一步讨论了每种风格各自的元素、关系、约束和质量属性等内容。最后，作者以一段简短总结作为结尾，再次强调了它们在架构设计中的核心作用。

2.知识梳理



3.体会感悟

作为这一章的核心内容，架构风格在软件的实际开发过程中不仅为设计提供了一致性和可预测性，同时还促进了团队间的有效沟通。“预制约束集”这一概念让我印象深刻，虽然约束可能会限制设计的灵活性，但同时也为系统的质量属性提供了保障。此外，通过对柏拉图式与体验式风格之间的对比，我认识到理想化的模型在实际应用中往往需要根据实际情况进行调整。这种灵活性和对现实的适应性是软件开发中不可或缺的。最后，通过学习不同的架构风格，我更加理解了如何在设计中做出合理的权衡，选择最合适的架构风格以满足系统的质量要求和业务需求。

4.重要观点/语句摘录

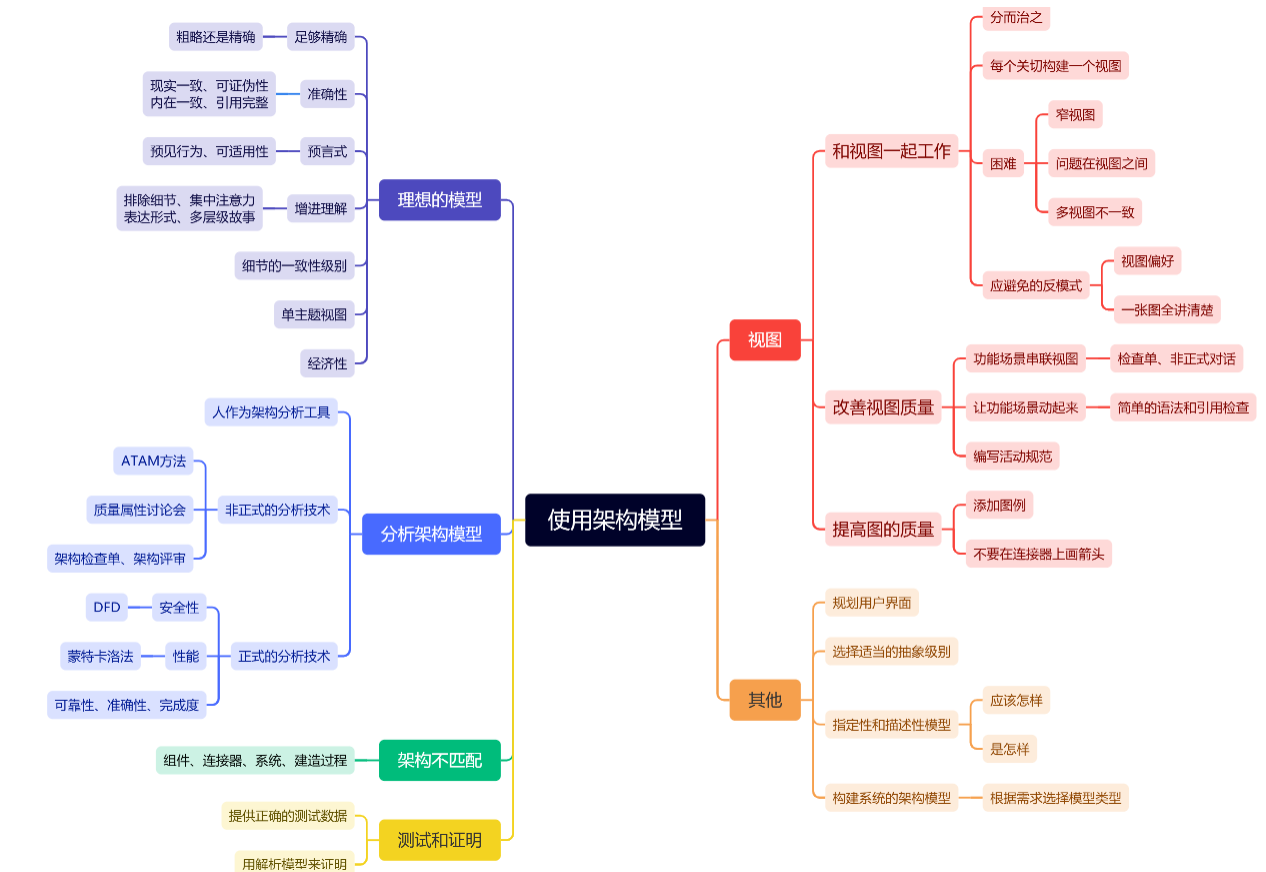
* “架构风格定义了一种由元素和约束组成的语言。”
* “风格约束带来的一致性可以促进系统干净地演化，这也使维护变得更简单了。”
* “以架构为中心的设计意味着你正在依靠架构来降低风险，实现某些特性或确保质量。”
* “柏拉图式架构风格是一种理想化，即柏拉图式的理想，像完美的圆。”
* “可以这么讲，大泥球风格就是没有任何清楚的结构，或者是一个坏结构的遗迹。”
* “分发-订阅风格最大的好处是，对事件的生产者和消费者进行了解耦。”
* “你可以把风格当做预制约束集，这个约束集既有优点也有缺点。如同任何一种预制品一样，你为自己节省了设计和调试的工作。预制品也许不能完全被调整到符合你的需要，但它的优点是，已经做好了，而且特性是已知的。”

**第十五章 使用架构模型**

1.章节回顾

这一章首先介绍了理想的模型应具备的精度、准确度、预测能力等关键特性，随后探讨了使用视图应遵循的思想，比如分而治之、聚焦单一关切等，并指出在使用视图时可能遇到的问题，如窄焦点、多视图的不一致性。为了解决这些问题，作者介绍了改善视图质量的方法，包括功能场景串联和编写活动规范，章节还探讨了分析架构模型，解释了架构不匹配的原因，介绍了相应的分析技术和分析工具。最后，围绕着一个现存系统，详细分析了建模过程中的挑战以及应该遵循的原则。整体而言，本章就使用架构模型提供了一套全面的指导。

2.知识梳理



3.体会感悟

架构模型是工程师思考和沟通的工具，其特性应该根据问题的需求来定，而不是一味追求完美。这让我想到实际工作中，我们经常因为追求过度的精确度而浪费时间，一个优秀的模型应在精确性和成本之间找到平衡点。书中草图和蓝图的类比，形象说明了在不同情况下选择合适的模型精确度的重要性。此外，视图的一致性和整体性对于理解复杂系统至关重要，可以通过功能场景和活动规范来改善视图质量。在架构分析中直觉和经验是不可或缺的，因此人被视为架构分析工具。在设计初期，就应该考虑到架构匹配问题，以免出现不同系统间的集成困难。

4.重要观点/语句摘录

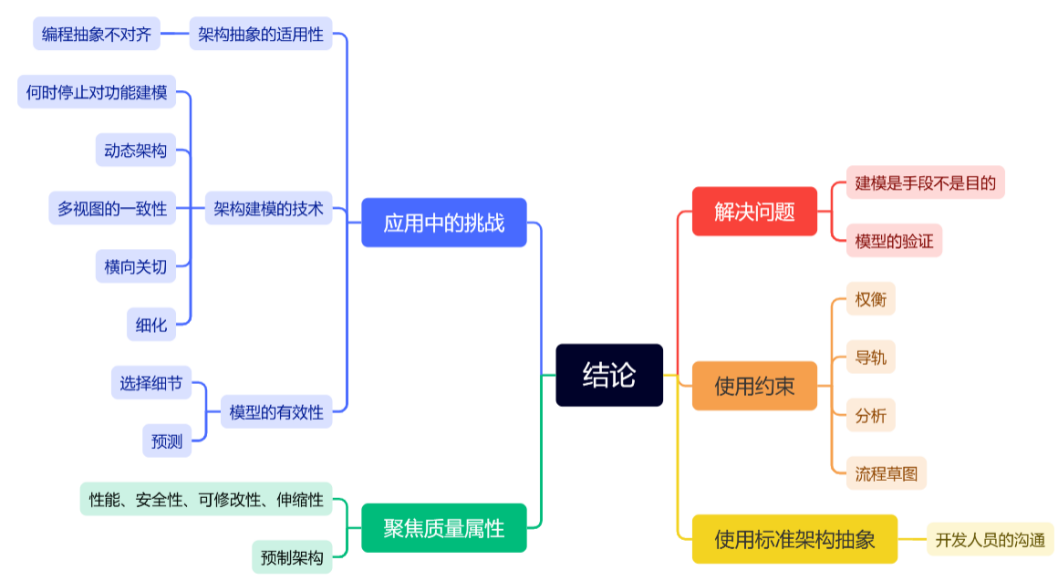
* “决定模型应该具备怎样的精确度的最好方法就是, 事先想好模型必须帮助你回答哪些问题。”
* “要让一个模型更好地被理解，最显著的方法是排除细节。”
* “作为工程师，你必须在构建模型的成本和模型带来的好处之间取得平衡。平衡的结果，通常是作出一个折中的、不那么完整的, 但是有用的模型。”
* “让场景动起来，意味着开发人员得遍历场景，让它一步一步在心里动起来。”
* “有着更多细节的模型并不总是比有着较少细节的模型好。事实上，模型应该力争实现细节的一致性级别，而不是追求某些部分比其他部分更详细。”
* “最好、最有用的模型都是有意识地对信息进行取舍后得到的。”
* “功能场景把隔离在各个视图中的内容装配成一个整体。”

**第十六章 结论**

1.章节回顾

这一章首先从应用软件架构和风险驱动模型时所面临的挑战开始，介绍了架构抽象的适用性、建模技术及模型的有效性。接着，聚焦于质量属性的重要性，并探讨了如何通过预制架构来满足特定领域的质量属性需求。然后，章节强调了解决问题的导向性，即不要仅仅停留在建模层面，并用一个幽默的灭火故事来强调模型验证的必要性。此外，章节还讨论了使用架构约束来实现预期结果的策略，包括权衡、导轨、分析和流程草图等方面。最后，章节强调了使用标准架构抽象在沟通和理解大型系统设计中的重要性。

2.知识梳理



3.体会感悟

架构设计不仅仅是技术层面的工作，更是一种战略层面的思考。特别是在质量属性中提到的，架构很大程度上决定了质量属性的实现难易，这让我意识到在项目初期就需要对质量属性给予足够的重视。此外，模型是帮助我们解决问题的工具，而绝不是目的本身，不能仅仅满足于构建模型，而应该将重点放在如何将模型转化为能够解决实际问题的系统上。此外，统一的沟通语言对于大型项目的成功至关重要，它能够帮助团队成员更有效地协作和理解彼此的工作。

4.重要观点/语句摘录

* “软件架构鼓励我们关注质量属性。 一般的软件开发人员通常会更多地关注系统功能，而不是质量属性或超功能需求。但是，架构在很大程度上决定着哪些质量是容易实现的，哪些是比较困难的，所以，在选择系统架构时，你的确应该更多地关注质量属性”
* “你也许倾向于认为，软件设计好了，问题就解决了，但事实是，只有当你在真实系统中构建出原型或演示产品时，才能确保模型得到验证。”
* “我们的目标是构建一个解决问题的系统，而不是构建模型。”
* “约束并不总是（或应该不是）任意的、变化无常的限制，导轨可以确保系统按照你指定的方向前进。”
* “架构抽象就像工具箱里的一个新工具，开发人员应该会在与规模和复杂度战斗时用到。”
* “如果开发人员不能利用架构抽象来思考，这个整体设计就呈现不出来，问题也就不会变得明显。”