

**SICHUAN UNIVERSITY**

****

**题 目** **《软件架构基础》读书笔记**

**学 院 软件学院**

**学生姓名 曹子杰**

**专 业 软件工程**

**学 号 2022141461049 年级 2022级**

**指导教师 毌攀良**

**2024年12月29日**

鉴于本学期我同时还选修了 软件设计与体系结构 这一门课程，我选择了《软件架构基础》这一本书作为我的阅读书籍，《软件架构基础》（*Fundamentals of Software Architecture*）是由Mark Richards和Neal Ford共同编写的作品，Mark Richards和Neal ford是专业教授软件架构课程多年的实践者，他们专注于应用于所有技术栈的架构原则。阅读本书，将从现代的角度探索软件架构，考虑过去十年的所有创新。

鹦鹉的修图

中度可信度描述已自动生成

以下是我阅读本书而作的读书笔记，整理了书中核心观点和实用知识点

介绍

首先是介绍了什么是软件架构，本书关于架构的定义主要从四个方面：

系统的结构（Structure）：**系统的结构指的是系统实现架构风格的类型（如微服务、分层或微内核）。但仅仅通过结构来描述一个架构，并不能完全阐明一个架构。**

系统所支持的架构特性、能力（Architecture characteristics）：**（架构特性多以 “*-ility*“ 结尾）**在设计架构时，需考虑各种非功能性需求（NFR），如性能、可扩展性、可维护性和安全性。

**关键架构特性：**

1. **性能（Performance）：** 系统处理请求的速度，通常与硬件资源和软件优化有关。
2. **可扩展性（Scalability）：** 系统在负载增加时扩展资源的能力。
3. **可用性（Availability）：** 系统在规定时间内正常工作的能力。
4. **可维护性（Maintainability）：** 系统的代码易读性和可修改性。
5. **安全性（Security）：** 防止系统受到攻击或数据泄露。

架构决策（Architecture decisions）：**架构决策定义了系统应该如何构建的规则。**

Mark Richards提出了架构设计是通过一系列权衡决策实现的：

* **决策的核心：** 确定哪些需求对业务最重要，然后优化架构来满足这些需求。
* **避免“银弹”：** 没有一种架构模式适合所有场景，选择架构时要基于实际需求。

**决策框架：**

1. 确定目标：明确系统的优先需求（如高性能、低延迟）。
2. 评估约束：识别技术和业务上的限制。
3. 比较选项：分析架构模式的优缺点，选择适合的解决方案。

**常见决策领域：**

* **数据库：** 关系型数据库（如MySQL） vs. 非关系型数据库（如MongoDB）。
* **通信：** 同步通信（REST API） vs. 异步通信（消息队列）。
* **部署：** 单体应用 vs. 容器化 vs. 云原生。

**例如，架构师可能会做出一个架构决策，即在分层架构中只有业务层和服务层可以访问数据库（见图），限制表现层直接调用数据库。架构决策形成了系统的约束，并指导开发团队什么是允许的，什么是不允许的。**

设计原则（Design principles） ：**设计原则与架构决策的不同之处在于，设计原则是一个指导方针，而不是一个硬性规定。例如，图示的设计原则指出，开发团队应该在微服务架构中的服务之间传递异步消息来提高性能。一个架构决策永远不可能涵盖服务之间通信的每一个条件和选项，设计原则为首选方法（在本例中，异步消息传递）提供指导，允许开发人员在特定情况下选择更合适的通信协议（如 REST 或 gRPC）。**

然后两位作者阐述了对 作为“架构师”的八个期待：

做出架构决定

架构师需要制定架构和设计原则，指导团队进行技术选择。重要的是提供方向而非具体工具，必要时确保关键架构特性的实现，并找到设计界线的平衡。

持续分析架构

架构师需定期评估架构的有效性，防止随着时间的推移发生结构性衰变。同时，需关注测试和发布环境的效率，以支持整体的敏捷性。

紧跟最新趋势

架构师应掌握最新的技术与行业动态，以支持长期决策的可行性。这需要他们不断学习和关注关键的技术趋势。

确保遵守各项决定

架构师需验证团队是否遵守既定的架构原则，防止因违规行为破坏系统特性。通过控制变化，确保系统的可维护性和可扩展性。

多样化的接触和经验

架构师应广泛接触不同技术、框架和平台，扩展技术广度以适应异构环境的需求。这种多样性经验有助于应对复杂的系统集成和跨平台问题。

具备业务领域知识

架构师设计有效解决方案的基础是理解业务问题和需求。优秀的架构师兼具技术深度与领域知识，能够用业务语言与利益相关者沟通。

具备人际交往能力

架构师需要具备卓越的团队合作、领导和沟通能力，以指导团队落实架构方案，解决技术与人之间的问题。

理解和驾驭企业政治

架构师需具备政治敏感性，能够处理利益相关者的挑战和意见分歧，并通过有效的谈判和沟通，为架构决策争取支持。

作为本学期多门选修课程的组长，包括：软件设计与体系结构 这一门课程小组的组长，也是软件项目管理小组的组长，我对于这一部分有一些自己的理解

例如对于最新趋势，我们在课上被要求学习了最新的软件设计趋势和方法，并制作自己的ppt进行了讲解，并且软件架构不是一成不变的，随着业务需求和技术环境的变化，架构需要不断演进。

**像是演进式架构（Evolutionary Architecture）**强调架构的可适应性和可扩展性。通过自动化测试和持续集成，支持小步快跑式的演进。

**也可以采取重构：**

1. **分阶段拆分：** 逐步将单体应用拆分为微服务。
2. **模块化：** 引入模块化设计减少耦合。
3. **技术更新：** 替换过时的技术栈，例如从传统服务器迁移到云环境。

具备人际交往能力也是，我们软件项目管理小组由九个人组成，需要交流如何进行小组任务分配，将人员的特质进行区分，并分配合理的任务是一门学问（这一点在团队激励与沟通这一门课程中，我也有所体会，成功的将知识融会贯通了）

可以说，优秀的架构不仅需要设计得当，还要清晰地表达给团队和利益相关者。

**文档化的原则：**

* **简洁明了：** 避免堆砌不必要的细节。
* **以读者为中心：** 针对技术团队、产品经理等不同目标受众调整内容。

**常见工具和格式：**

* **C4模型：** 用于描述上下文、容器、组件和代码的关系。
* **架构决策记录（ADR）：** 文档化关键决策及其背景和影响。

**沟通技巧：**

* 用视觉化图表（如流程图）解释复杂概念。
* 确保与团队成员、产品负责人频繁对齐，避免信息孤岛。

并阐述了架构师和其他人员的交集，这里不做过多赘述。

然后谈及了软件架构法则，这是我认为比较重要的内容：

软件架构第一定律：

**软件架构中的所有东西都是一种权衡。(Everything in software architecture is a trade-off.)，**

**权衡示例：**

为了提高性能，可能会牺牲代码的易读性和灵活性。

分布式架构虽然可扩展，但需要投入更多的基础设施成本。

**要求高效率、高性能，那就不可避免地会提高软件复杂性，还可能降低可用性。**

**而要求高的可用性，例如全天24小时不允许出错，那就会注定的提高软件复杂性，降低效率（设计一系列安全检查机制的过程，大多会降低性能）**

我们对软件架构的定义超越了结构的范畴，包含了原则、特性等，架构的范围比单纯的结构更广，体现在我们的软件架构第二定律中：**为什么比怎么做更重要。（Why is more important than how.），不只是软件架构，学习绝大多数知识，都要明白为什么，尤其是设计部分，不然就是生搬硬套**

架构思维

**架构思维指用架构的眼光和观点来看待事物，主要包括：**

* **理解软件架构和软件设计的区别，知道与开发团队合作，并让架构发挥作用。**
* **拥有技术广度的同时，保持一定的技术深度，看到别人看不到的解决方案和可能性。**
* **理解、分析、协调各种解决方案和技术之间的权衡。**
* **理解业务驱动的重要性，以及如何将其转化为架构**

**架构师应与开发团队之间形成双向关系，架构师分析业务需求，提取定义架构特性后，选择架构风格，以及创建组件，然后交给开发团队，而开发团队为每个组件创建类、用户界面，以及开发和测试源码后应当有所反馈，并进行下一次迭代，当今系统的架构在项目的每一次迭代都会发生变化和发展，架构师和开发团队之间的紧密合作是任何软件项目成功的关键。 并且架构师应当参与编程任务但作为非全职开发人员，尽量避免掌握关键路径和代码框架，应将其委托给团队中的其他人，然后集中精力在迭代后对一个业务功能进行编码，处理一些简单的重复任务，并做code review**

**架构模式和风格**

书中还探讨了各种常见的架构模式及其适用场景：

**单体架构（Monolithic Architecture）**

* 所有功能模块都打包在一个应用中。
* **优点：** 简单、部署方便。
* **缺点：** 随着规模扩大，难以维护和扩展。

**微服务架构（Microservices Architecture）**

* 将系统拆分为多个独立服务，每个服务独立部署和运行。
* **优点：** 易于扩展和独立部署，适合复杂系统。
* **缺点：** 增加分布式系统的复杂性，例如通信和数据一致性问题。

**事件驱动架构（Event-Driven Architecture）**

* 系统通过事件异步通信，松耦合各部分。
* **优点：** 具有高度的可扩展性和灵活性。
* **缺点：** 调试和测试较为复杂，可能引入意外的副作用。

**分层架构（Layered Architecture）**

* 系统划分为逻辑层（如表示层、业务逻辑层和数据访问层）。
* **优点：** 易于理解，适合初学者和中小型系统。
* **缺点：** 层级之间可能形成紧耦合，降低灵活性。

**面向服务的架构（SOA）**

* 强调服务重用和集成，通常基于企业服务总线（ESB）。
* **与微服务的区别：** SOA服务粒度较粗，技术依赖性较强，而微服务强调独立性。

**总结**

《软件架构基础》一书全面系统地分析了软件架构领域的重要概念、方法和实践建议。书中通过具体的案例和理论相结合，强调了架构设计的艺术性和技术性。通过本书的学习，读者不仅能够掌握架构设计的核心理念，还能在实际项目中应用这些知识，设计出高效、可靠、可扩展的系统。