Chapter 1. What is Software Architecture?

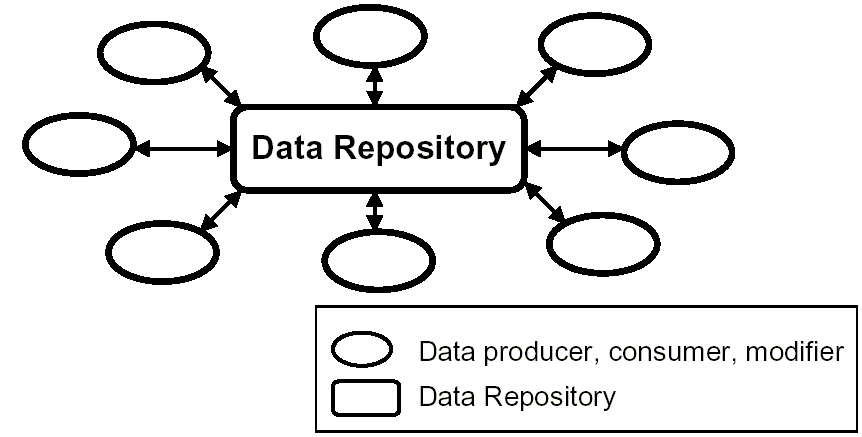
【理解：软件体系结构（软件架构）的定义、架构模式的概念】

**What is Software Architecture?**

***The software architecture of a system is the set of structures needed to reason about the system, which comprise software elements, relations among them, and properties of both****.*

**Architectural Patterns**

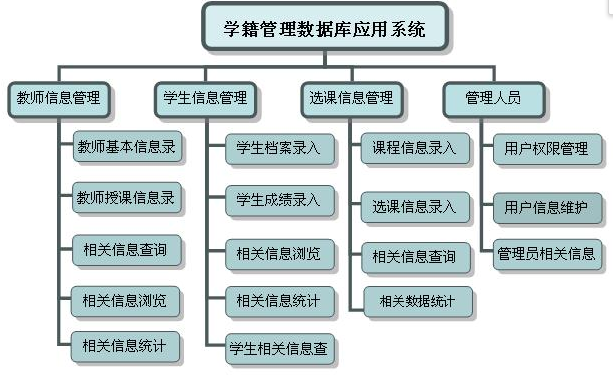
* 架构元素的组合称为架构模式。
  + 模式为解决系统面临的一些问题提供了打包的策略。
* 架构模式描述元素类型及其交互形式。
* 常见的模块类型模式是分层模式。
* 常见的组件和连接器类型模式:
  + 数据共享模式（Shared-data pattern）



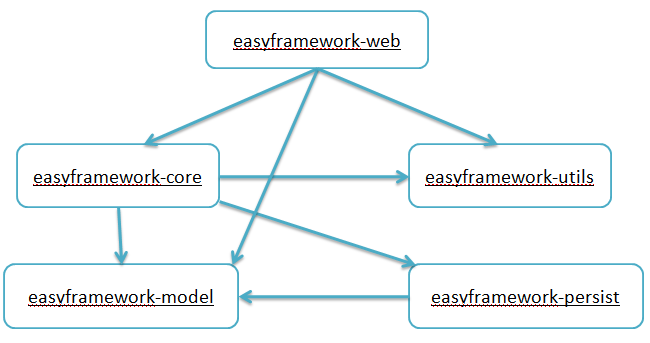
* + 客户端-服务器模式（Client-server pattern）
    - 组件是客户机和服务器。
    - 连接器是协议和消息，它们彼此共享以执行系统的工作。
* 常见的分配模式:
  + 多层模式（Multi-tier pattern）
  + 能力中心模式和平台模式（Competence center pattern and platform pattern）
    - 这些模式专门化了软件系统的工作分配结构。
    - 在能力中心，工作是根据能力来分配的。
    - 在平台中，一个站点负责开发软件产品线的可重用核心资产，而其他站点则开发使用核心资产的应用程序。

【掌握：软件系统有哪几类结构？在每类结构里，元素及其之间的关系是什么？每类结构各有哪些常见的结构？其特点是什么？】

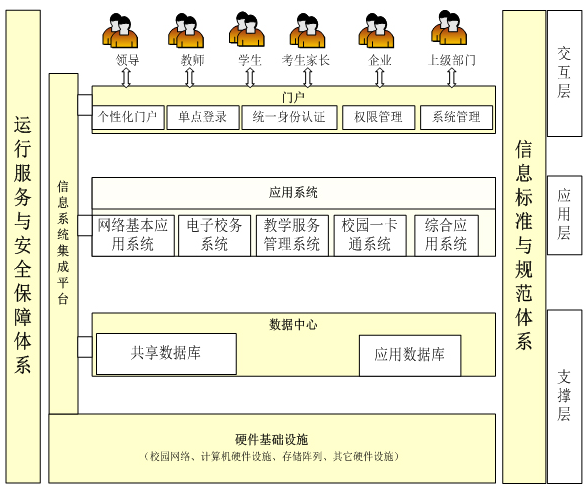
* Software systems are composed of many structures.
  + Module
  + Component and Connector
  + Allocation
* Some Useful Module Structures
  + 分解结构（Decomposition structure）
    - 单元是通过is-a-sub - module关系相互关联的模块。
    - 分解结构通过确保可能的更改是本地化的，从而确定系统的可修改性。
    - 这种结构通常用作开发项目组织的基础，包括文档、集成和测试计划。



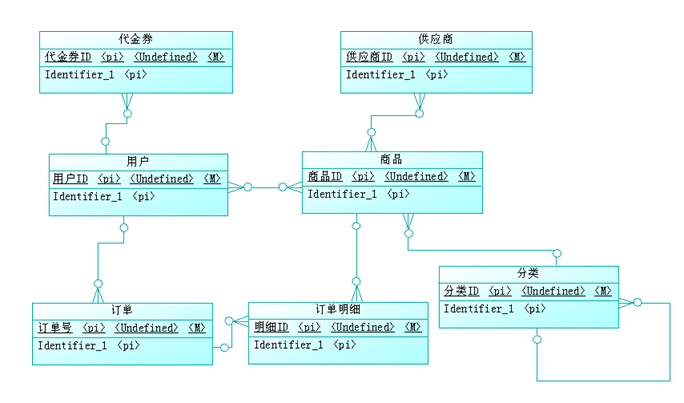
* + 使用结构（Uses structure）
    - 这些单元通过使用关系联系在一起，使用关系是依赖关系的一种特殊形式。
    - 如果第一个单元的正确性要求存在第二个单元的正确版本，则软件的一个单元使用另一个单元。
    - 使用结构用于扩展或契约系统。
    - 轻松创建系统子集的能力允许增量开发。



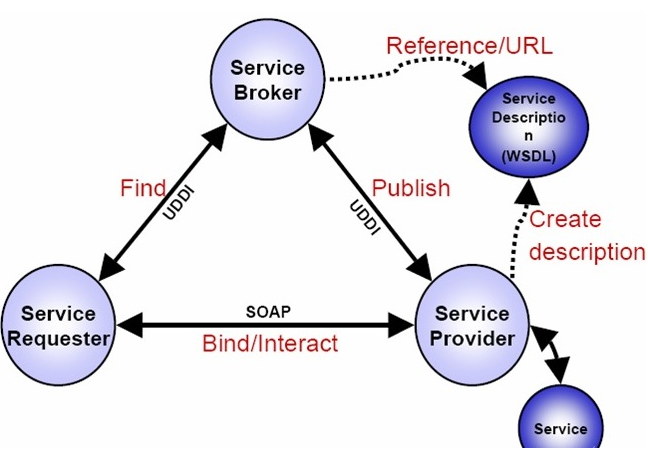
* + 层结构（Layer structures）
    - 这种结构中的模块称为层。
    - 层是一个抽象的“虚拟机”，它通过托管接口提供一组内聚的服务。



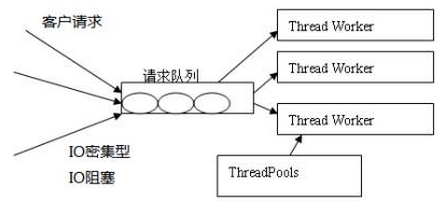
* + 类(或泛化)结构（Class (or generalization) structures）
    - 这种结构中的模块单元称为类。
    - 该关系继承自、是或组成的实例。
    - 这些视图支持关于类似行为或功能集合的推理
      * 例如，其他类继承的类和参数化的区别
    - 类结构允许考虑重用和功能的增量添加。
  + 数据模型（Data model）
    - 数据模型根据数据实体及其关系描述静态信息结构。



* Some Useful C&C Structures
  + 服务结构（Service structure）
    - 这些单元是通过服务协调机制(如SOAP)相互操作的服务。



* + 并发性结构（Concurrency structure）
    - 这种结构有助于确定并行性的机会以及可能发生资源争用的位置。



* Some Useful Allocation Structures
  + 部署结构（Deployment structure）
    - 部署结构显示了如何将软件分配给硬件处理和通信元素。
    - 这些元素是软件元素(通常是C&C视图中的流程)、硬件实体(处理器)和通信路径。
    - 如果分配是动态的，则关系被分配到，显示软件元素驻留在哪个物理单元上，并迁移到。
    - 此结构可用于推断性能、数据完整性、安全性和可用性。
    - 它对分布式和并行系统特别感兴趣。
  + 实现结构（Implementation structure）
    - 该结构显示了软件元素(通常是模块)如何映射到系统开发、集成或配置控制环境中的文件结构。
  + 工作分配结构（Work assignment structure）
    - 该结构将实现和集成模块的责任分配给将执行模块的团队。
* Structures Provide Insight
  + 模块结构包含了模块使用的模块和其他模块，它与系统扩展或收缩的易用性密切相关。
  + 并发结构体现了系统内的并行性，它与使系统摆脱死锁和性能瓶颈的易用性密切相关。
  + 部署结构与性能、可用性和安全目标的实现紧密相关。

【了解：结构与视图是什么关系？好的结构的一些经验法则。】

* Structures and Views
  + 视图是结构的表示。
    - 例如，模块结构是系统模块及其组织的集合。
    - 模块视图是该结构的表示，由一些系统涉众记录并使用。
  + 架构的设计结构。它们记录这些结构的视图。
* What Makes a “Good” Architecture?
  + 没有天生的好或坏架构。
  + 架构或多或少适合于某些用途
  + 架构可以被评估，但是只能在特定目标的上下文中。
  + 然而，经验法则也不错。
* Process “Rules of Thumb”
  + 体系结构应该是单个架构师的产品，或者是具有确定的技术领导的一小群架构师的产品。
  + 架构师(或体系结构团队)应该将体系结构建立在良好指定的质量属性需求的优先级列表上。
  + 应该使用视图记录体系结构。
  + 应该评估体系结构交付系统重要质量属性的能力。
  + 这应该发生在生命周期的早期，并在适当的时候重复。
  + 体系结构应该适合于增量实现。
* Structural “Rules of Thumb”
  + 体系结构应该具有定义良好的模块，这些模块的功能职责是根据信息隐藏和关注点分离的原则分配的。
  + 质量属性应该使用特定于每个属性的知名架构模式和策略来实现。
  + 体系结构永远不应该依赖于商业产品或工具的特定版本。
  + 产生数据的模块应该与使用数据的模块分开。
  + 应该编写每个进程，以便能够容易地更改其分配给特定处理器的任务，甚至在运行时也可能如此。
  + 该体系结构应该提供少量组件交互的方式。
  + 架构应该包含一组特定的(和少量的)资源争用区域。

【本章总结】

* 系统的软件体系结构是对系统进行推理所需要的一组结构，这些结构包括软件元素、它们之间的关系以及两者的属性。
* 结构是元素及其相互关系的集合。
* 视图是一组一致的架构元素的表示。视图是一个或多个结构的表示。
* 有三种类别的结构。
* 结构表示体系结构的主要工程杠杆点。
* 每个系统都有一个软件体系结构，但是这个体系结构可能是文档化和传播的，也可能不是。
* 没有天生的好或坏架构。架构或多或少适合于某些用途。