

# 软件工程

大连理工大学软件学院

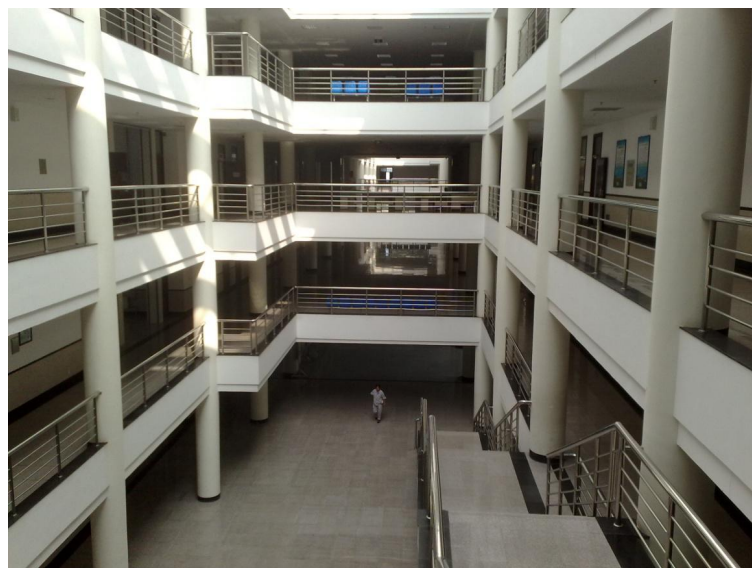


# 第4章 软件架构的构建

- 软件架构也称为软件体系结构。
- 对软件架构的系统、深入的研究将成为提高软件生产率和解决软件维护问题的新途径。
- 用户需求中的各种约束对架构的选择都有着直接的影响，软件开发并不是一种超脱的自由发挥。
- 本章主要介绍软件架构的基本概念、相关模型、风格及其设计方法。

# 系统架构

- 软件架构设计就是建立系统所需的数据结构和程序构件，考虑：
  - 体系结构风格
  - 组成构件的结构和属性
  - 所有体系结构构件之间的相互关系
- 如同土木工程，软件也从传统的软件工程进入现代面向对象的软件工程，研究整个软件系统的体系结构，并寻求建构最快、成本最低、质量最好的构造过程。

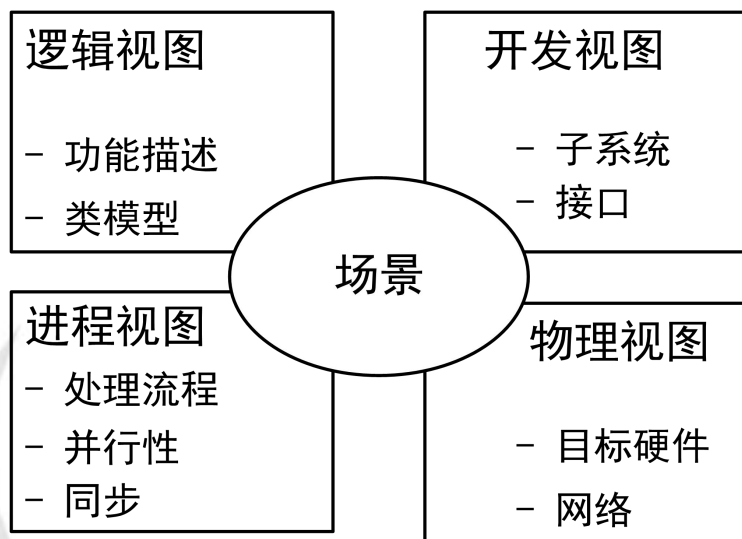


# 软件架构的通识定义

- 软件架构为软件系统提供了一个结构、行为和属性的高级抽象，由构成系统的元素的描述、这些元素的相互作用、指导元素集成的模式以及这些模式的约束组成。
- 软件架构不仅指定了系统的组织(Organization)结构和拓扑(Topology)结构，并且显示了系统需求和构成系统的元素之间的对应关系，提供了一些设计决策的基本原理。

# 软件架构的“4+1”视图模型

- 如何表示软件架构，即如何对软件架构建模。
- 根据建模的侧重点不同，可以将软件架构的模型分为五种：**结构模型、框架模型、动态模型、过程模型和功能模型**。



- “4+1”视图模型从五个不同的视角来描述软件架构。
- 每个视图只关心系统的一个侧面，五个视图结合在一起才能反映系统的软件架构的全部内容。

# 软件架构的基本元素


- **构件**是具有某种功能的可重用的软件模板单元，表示了系统中主要的计算元素和数据存储
  - 复合构件
  - 原子构件
- **连接件**表示构件之间的交互
  - 简单的连接件如：管道(Pipe)、过程调用(Procedure call)、事件广播(Event broadcast)等
  - 复杂的交互，如客户/服务器(Client/Server)通信协议、数据库和应用之间的SQL连接等。
- **配置**表示了构件和连接件的拓扑逻辑和约束

# 软件架构的风格

- 软件架构设计的一个核心问题是能否使用重复的体系结构模式，即能否达到体系结构级的软件重用。
- 软件架构风格是描述某一特定应用领域中系统组织方式的惯用模式。
- 体系结构风格定义了一个系统家族，包括体系结构的定义、词汇表和一组约束。软件架构风格定义了用于描述系统的术语表和一组指导构建系统的规则。
- 软件架构风格促进了对设计的重用，不变的部分使不同的系统可以共享同一实现代码，只要系统是使用常用的、规范的方法来组织，就可使别的设计师很容易地理解系统的体系结构。



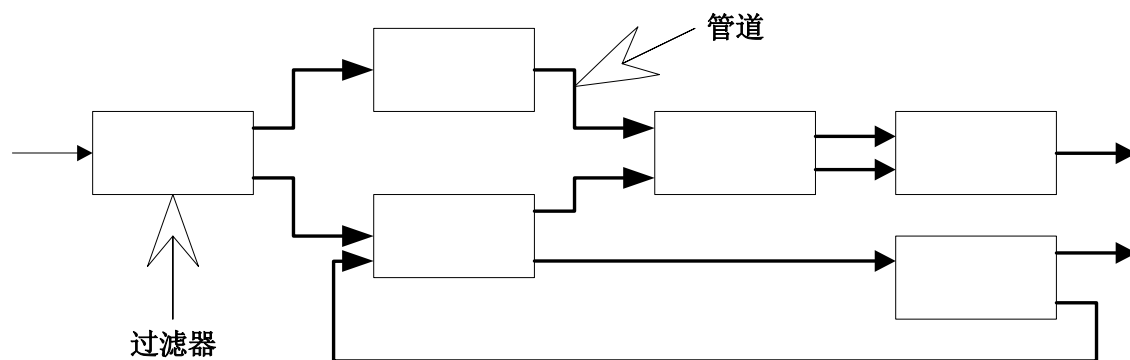
# 通用体系结构风格的分类

- 
1. 数据流风格：批处理序列、管道与过滤器等。
  2. 调用/返回风格：层次结构、正交软件结构、客户机/服务器结构、浏览器/服务器结构等。
  3. 独立构件风格：进程通信、事件系统、MVC结构等。
  4. 虚拟机风格：解释器、基于规则的系统等。
  5. 数据中心风格：数据库系统、超文本系统、仓库/黑板系统等。
- 本书介绍几种主要的软件架构风格，它们常用于主流系统的设计。



# 管道与过滤器

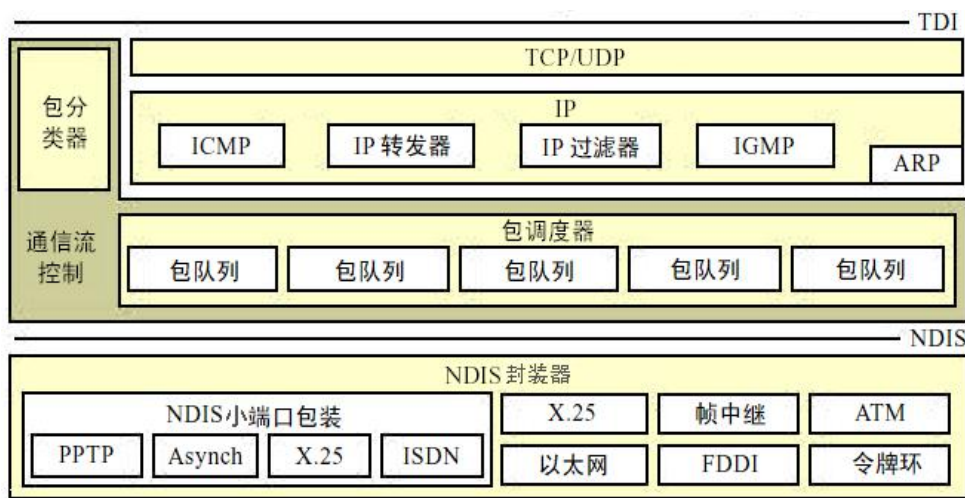
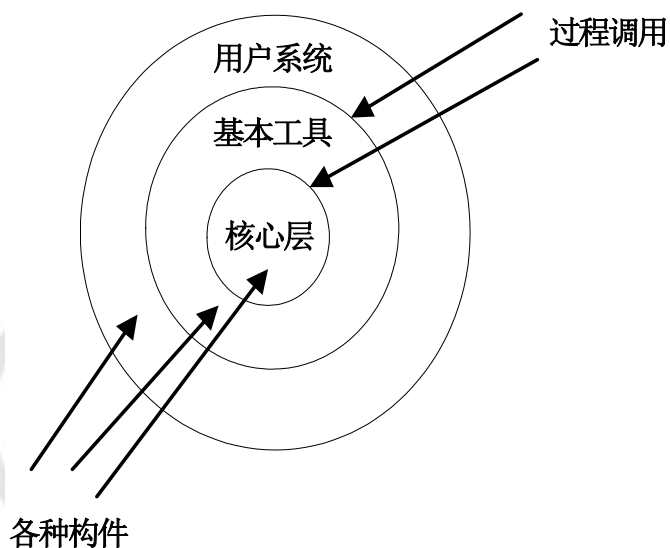
- 每个构件都有一组输入和输出，构件读取输入的数据流，经过内部处理，然后产生输出数据流。
- 构件被称为过滤器，这种风格的连接件就是数据流传输的管道，将一个过滤器的输出传到另一过滤器的输入。
- 过滤器是独立的实体，不能与其他的过滤器共享数据，而且一个过滤器不知道它上游和下游的标识。



- 适合批处理和非交互处理的系统，使软件具有良好的信息隐藏性和模块独立性，从而产生高内聚、低耦合的特点。

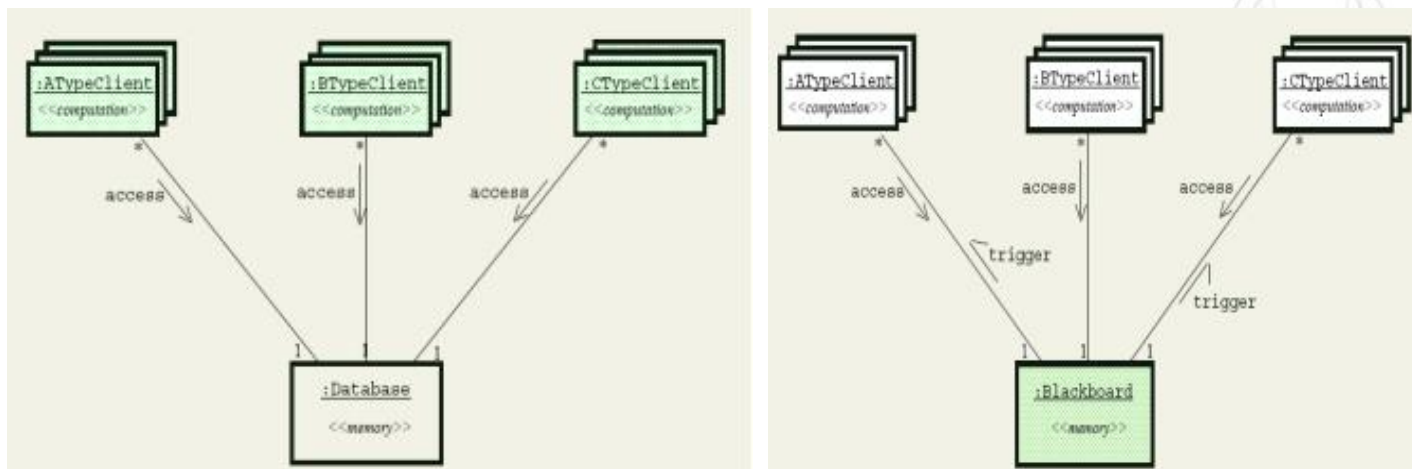
# 层次系统

- 层次系统中，每一层为上层提供服务，并作为其下层客户。
- 连接件通过决定层间如何交互的协议来定义，拓扑约束包括对相邻层间交互的约束。
- 这种风格支持基于可增加抽象层的设计，允许将一个复杂问题分解成一个增量步骤序列的实现。

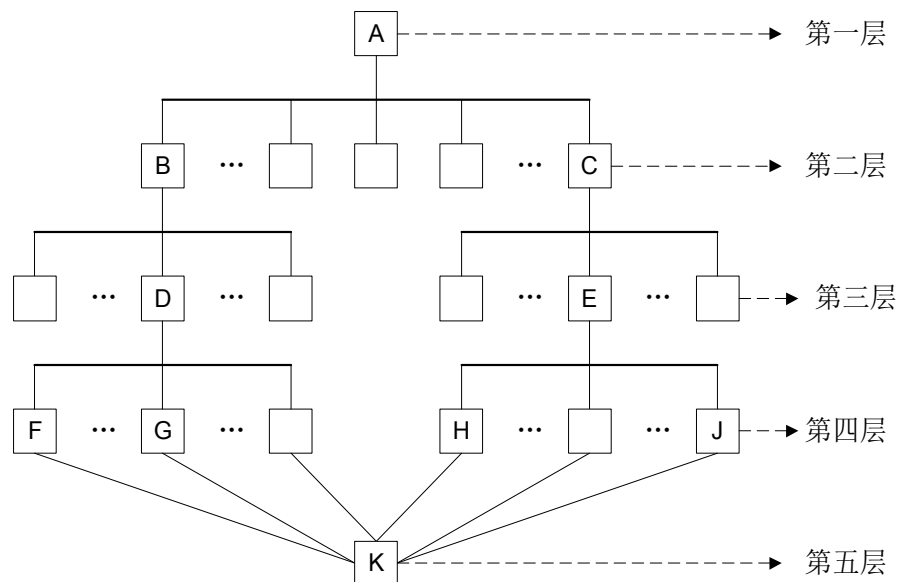


# 仓库系统

- 中央数据结构说明当前状态，独立构件在中央数据存储上执行，仓库与外构件间的相互作用在系统中会有较大的变化。
- 控制原则的选取将产生两个主要的子类：
  - 若输入流中某类时间触发进程执行选择，仓库是传统型数据库；
  - 若中央数据结构的当前状态触发进程执行选择，仓库是黑板系统。
- 黑板系统的传统应用是信号处理领域，如语音和模式识别，另外的应用包括松耦合代理数据共享存取等。



# 正交软件架构

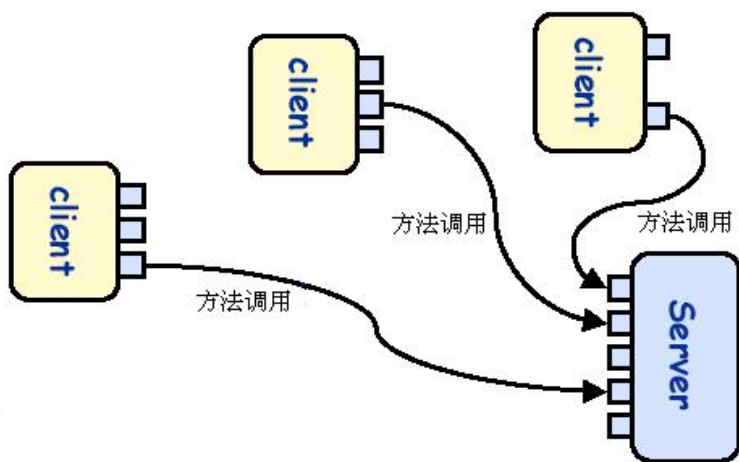


- 由组织层和线索的构件构成
  - 层是由一组具有相同抽象级别的构件构成;
  - 线索是子系统的特例，它由完成不同层次功能的构件组成，通过相互调用来关联，每一条线索完成整个系统中相对独立的一部分功能。

- 如果线索是相互独立的，即不同线索中的构件之间没有相互调用，那么这个结构就是完全正交的。
- 正交软件架构是一种以垂直线索构件族为基础的层次化结构。
- 在软件演化过程中，系统需求会不断发生变化。在正交软件架构中，因线索的正交性，每个需求变动仅影响某一条线索。

# 客户机/服务器架构

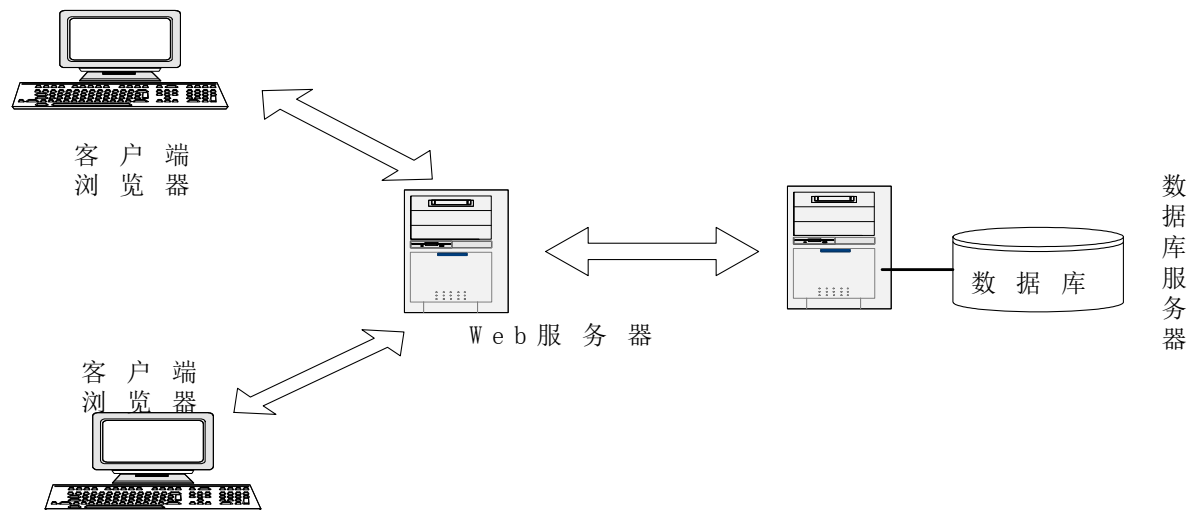
- 服务器负责管理系统资源：访问与并发性控制、安全性、备份与恢复和全局数据完整性规则。
- 客户应用程序提供用户与服务器交互的界面、向服务器提交用户请求并接收来自服务器的信息、利用客户应用程序对存在于客户端的数据执行应用逻辑要求。
- 网络通信软件的主要作用是完成服务器和客户应用程序之间的数据传输。



- 服务器为多个客户应用程序管理数据
- 对于硬件和软件的变化具有极大的适应性和灵活性
- 易于对系统进行扩充和缩小
- 系统中的功能构件充分隔离

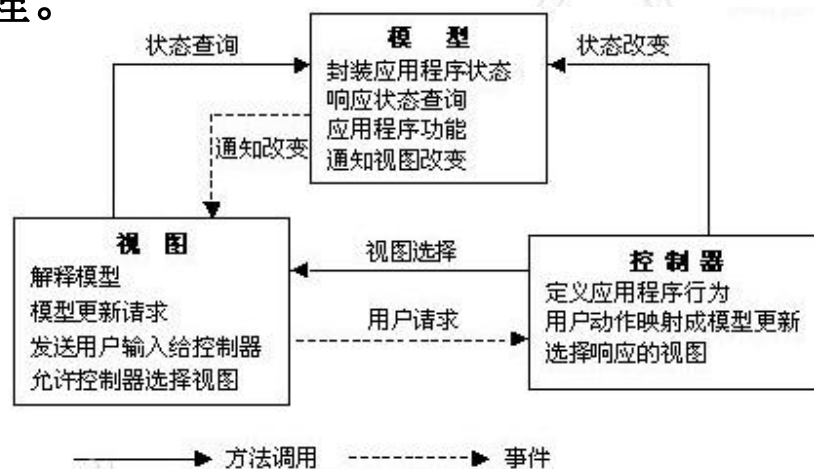
# 浏览器/服务器架构

- B/S是C/S的扩展，瘦客户。
- 应用(程序)在一定程度上具有集中特征。
- 减轻安装、配置和升级等维护工作。
- 层与层之间相互独立，任何一层的改变都不影响其他层原有的功能，所以可用不同厂家的产品组成性能更佳的系统。（平台透明性）



# MVC架构

- 模型(Model)－视图(View)－控制器(Controller)
- MVC是一种软件设计典范，用业务逻辑、数据、界面显示分离的方法组织代码，将业务逻辑聚集到一个部件里面，在改进和个性化定制界面及用户交互的同时，不需要重新编写业务逻辑。
- 视图是用户看到并与之交互的界面。
- 模型表示企业数据和业务规则。
- 控制器接受用户的输入并调用模型和视图去完成用户的需求，控制器本身不输出任何结果和做任何处理。

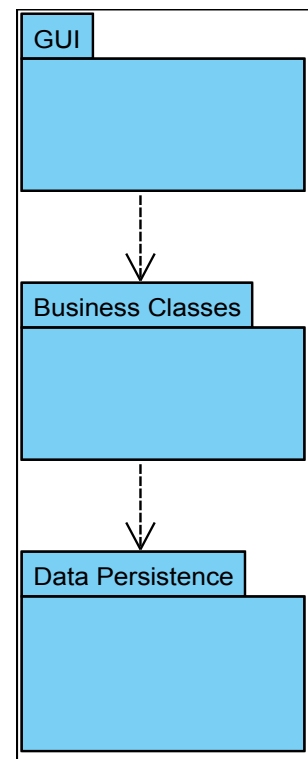
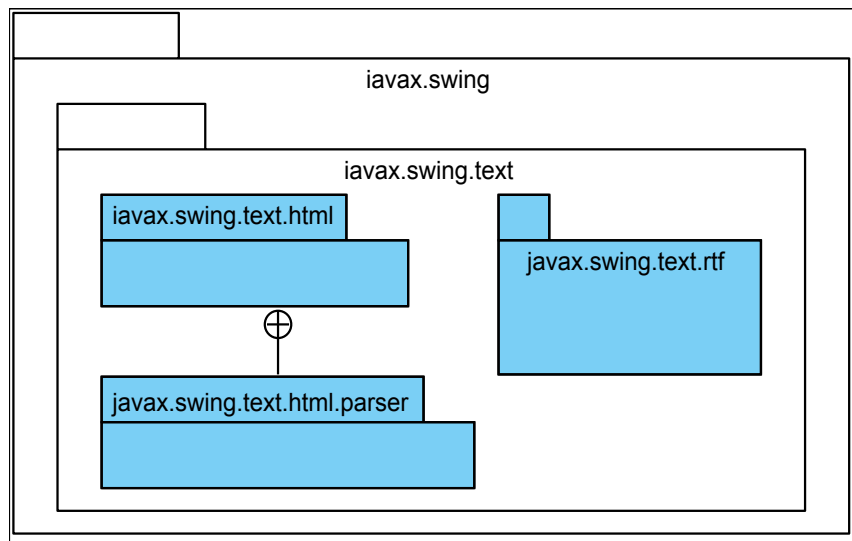




# 软件架构设计

- 在软件架构设计的起初，要考虑软件必须放在所处的环境。
- 一旦建立了软件的环境模型，并且描述出所有的外部软件接口，那么设计师就可以通过定义和求精实现架构的构件来描述系统的结构。这个过程不停地迭代，直到获得一个完善的架构。
- 通过软件架构，系统将逻辑关系密切的单元划分到一起，形成系统的逻辑划分，有利于后续独立的开发和管理。
- 这个划分经常是基于类模型进行的，并可参照一些设计优化方法形成更合理的组织方式，达到模块内部的高内聚和模块间的低耦合。
- 软件架构对应的实现就是将软件使用所谓的“包(Package)”进行构造，每个包对应某种专属的功能，并尽可能独立。
- 包中的类互相紧密配合协作完成包的功能，每个包与其他包中含有的类之间的接口应该尽可能简单，降低它们的耦合性。

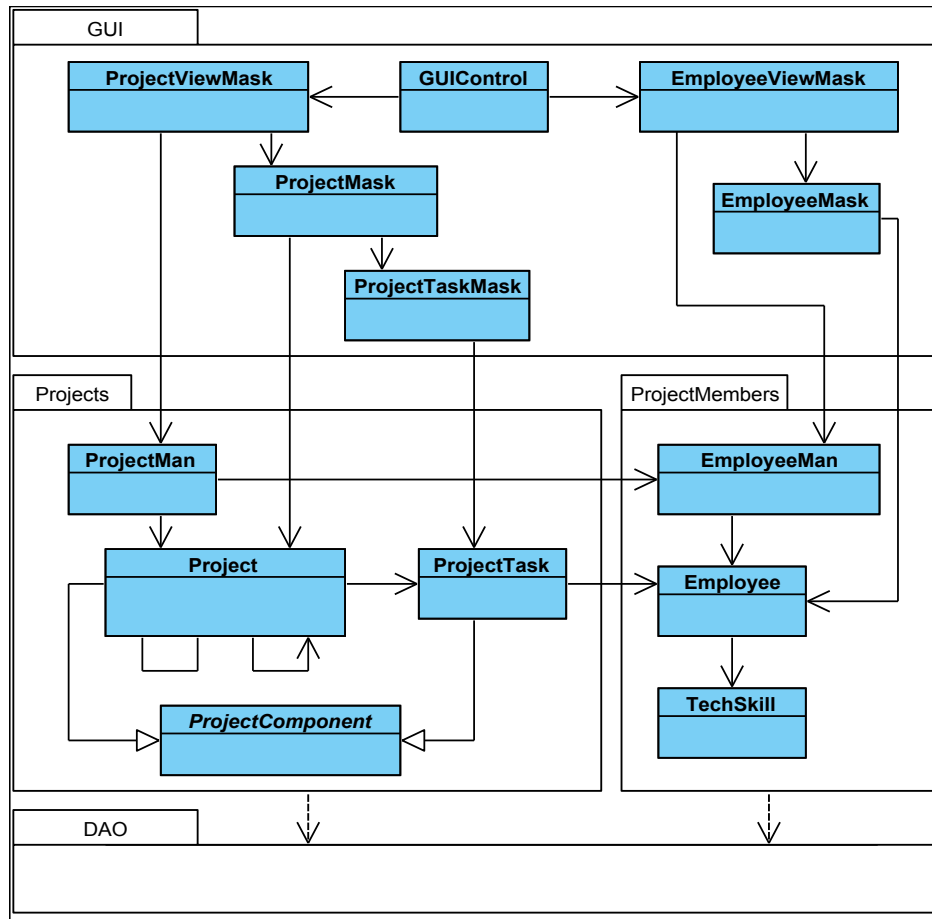
# 包及其结构



- 包之间的虚线箭头描述的是层间的使用(依赖)关系
- 包图中一个重要的要求是在包间不能出现循环的依赖
- 在循环依赖的情况下, 对于任何包中的修改, 都要对各个依赖方向上的包进行循环检查其影响范围, 这显然是非常繁琐和不可取的。

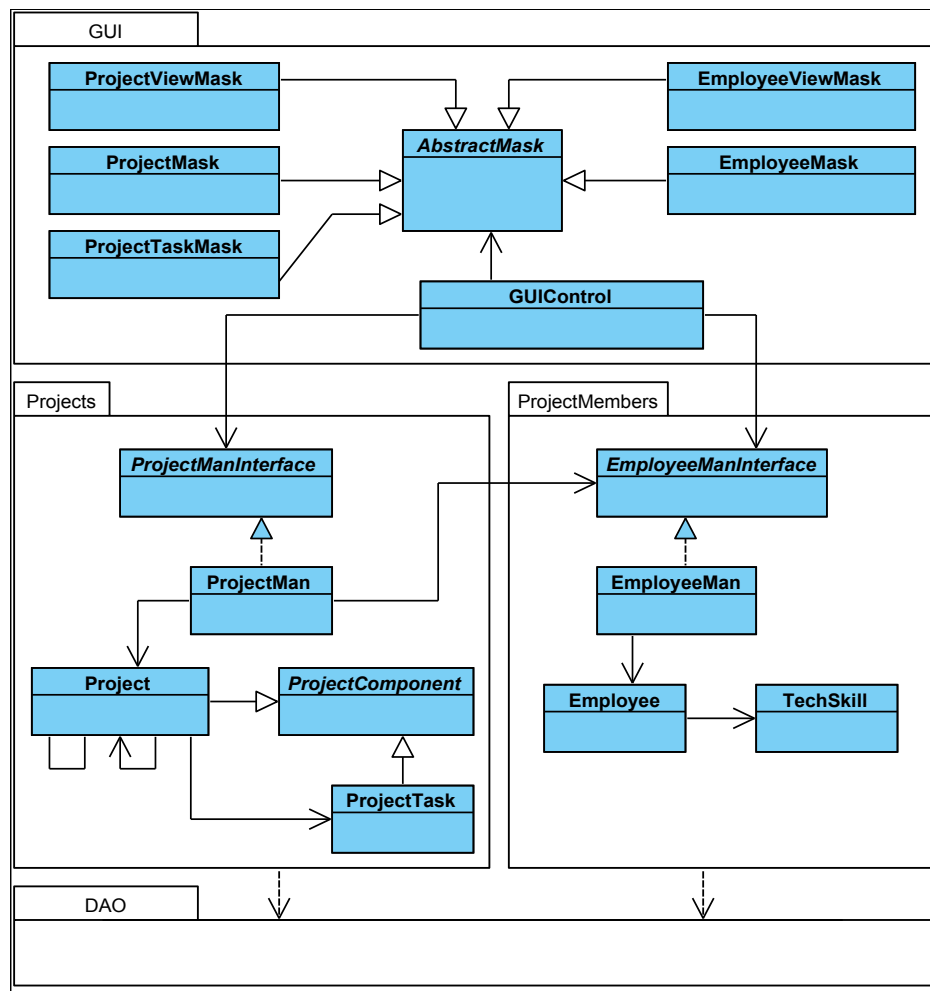
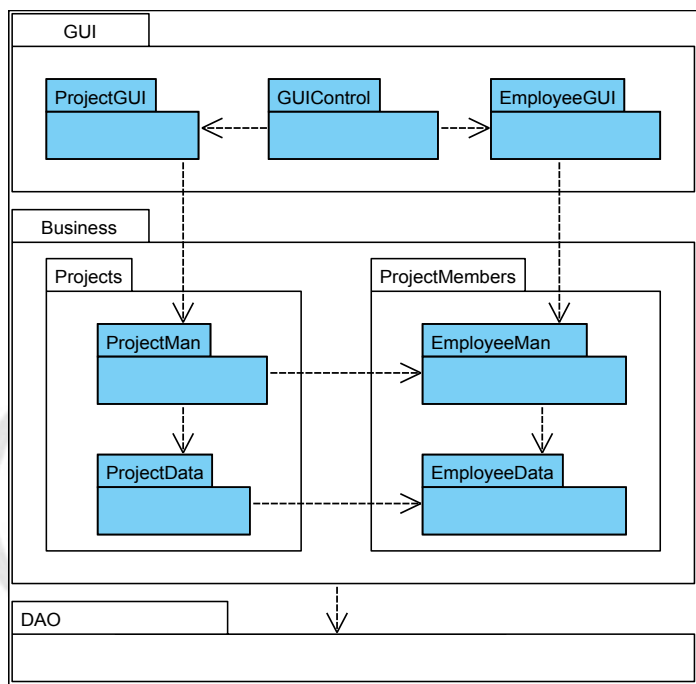
# 包结构设计

- 应尽量减少包中全局类的数量。
- 初始的项目管理系统的层次架构。
- 图形界面包(GUI)中设置了一个控制类GUIControl，其作用是将用户事件的处理向不同的业务接口类分发，同时也使得该架构有了MVC的雏形。
- Mask类，是按照对应的业务类设置的，其作用是提供额外的与显示和控制的相关信息，如数据校验规则等。



# 包结构设计

- 将类更加合理地在包间划分和组织，以提高各部分的独立性，从而达到更高的内聚性。
- 对包的宏观结构进行了调整，突出了整体的层次。



# 作业

- 习题1

