



# 系统架构设计师

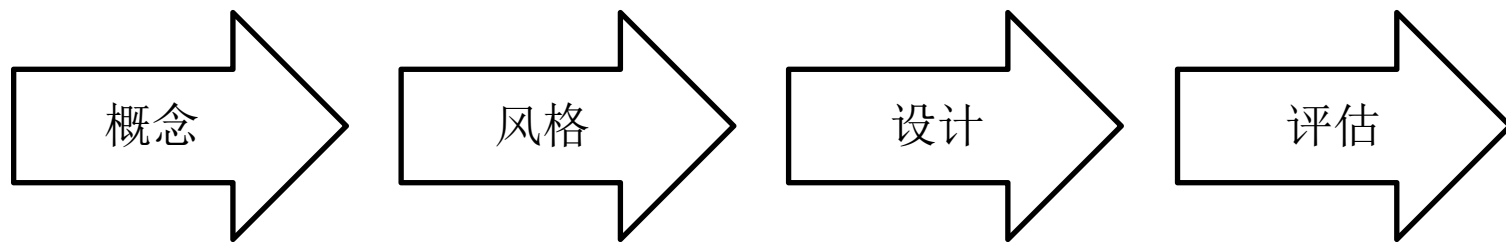
DESIGNER: 王川林  
软件架构设计



## 课程内容

- ◆ 软件架构的概念 ★ ★ ★
- ◆ 软件架构风格 ★ ★ ★ ★ ★
- ◆ 架构描述语言ADL ★ ★ ★
- ◆ 特定领域软件架构 ★ ★ ★
- ◆ 基于架构的软件开发方法 ★ ★ ★ ★
- ◆ 软件质量属性 ★ ★ ★ ★ ★
- ◆ 软考架构评估 ★ ★ ★ ★ ★
- ◆ 软件产品线 ★ ★ ★
- ◆ 中间件技术 ★ ★ ★
- ◆ 典型应用架构 ★ ★ ★



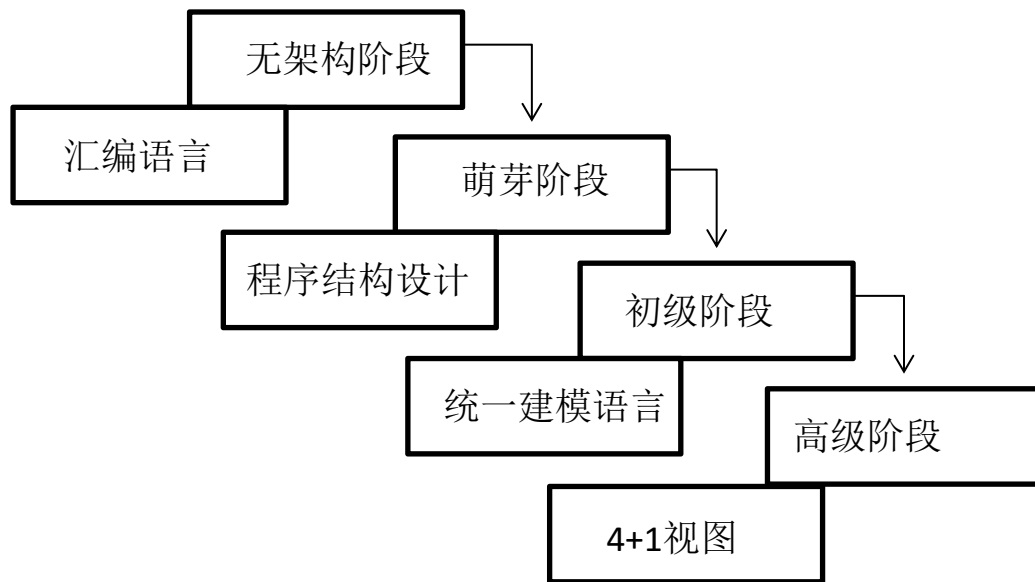


需求分析    架构    软件设计

鸿沟

架构设计就是需求分配，即将满足需求的职责分配到组件上

- 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。
- 软件架构的是项目干系人进行交流的手段，明确了对系统实现的约束条件，决定了开发和维护组织的组织结构，制约着系统的质量属性
- 软件架构使推理和控制的更改更加简单，有助于循序渐进的原型设计，可以作为培训的基础
- 软件架构是可传递和可复用的模型，通过研究软件架构可能预测软件质量。
- 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。



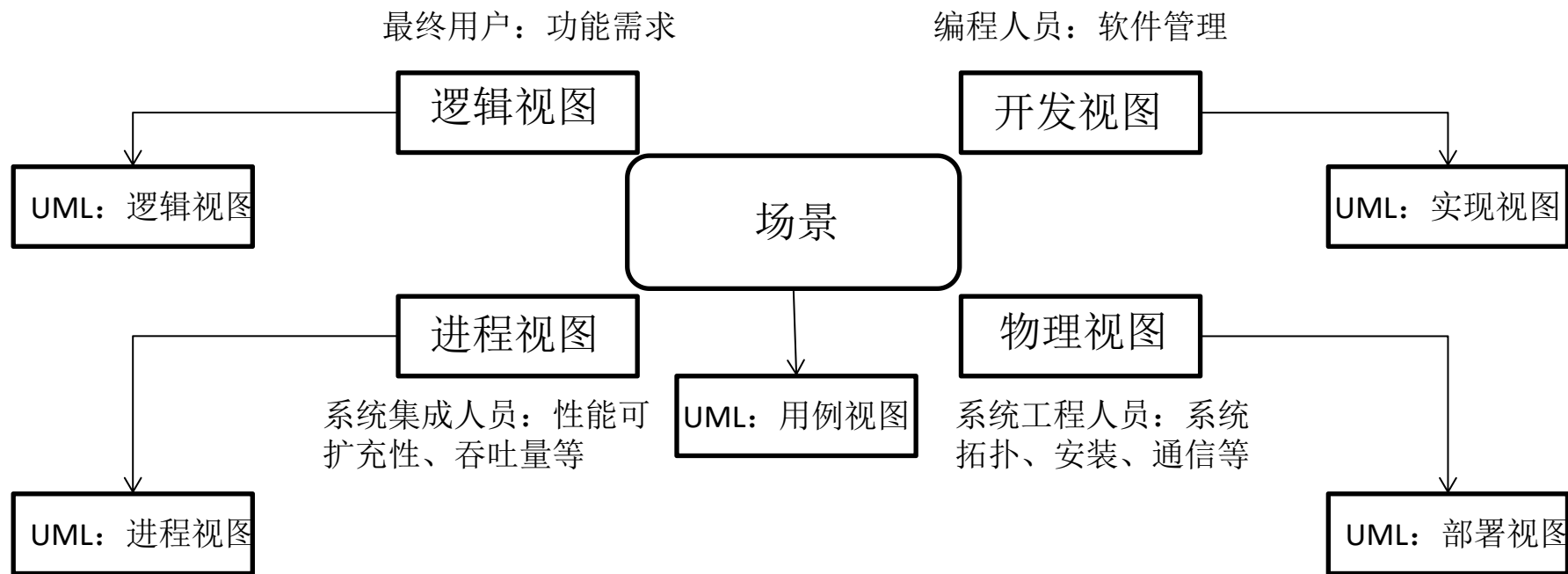
结构模型：以架构的构件、连接件和其他概念来刻画结构

框架模型：不太侧重描述结构的细节而更侧重于整体的结构

动态模型：系统的“大颗粒”的行为性质

过程模型：构建系统的步骤和过程

功能模型：由一组功能构件按层次组成，下层向上层提供服务



体系结构描述语言（architecture description language，简称ADL），支持构件、连接件及其配置的描述语言就是如今所说的体系结构描述语言。ADL对连接子的重视成为区分ADL和其他建模语言的重要特征之一。典型的ADL包括Unicon，Rapide，Darwin，Wright，C2 SADL，Acme，xADL，XYZ/ADL，ABC/ADL等



- 架构设计的一个核心问题是能否达到架构级的软件复用
- 架构风格反映了领域中众多系统所共有的结构和语义特性，并指导如何将各个构件有效地组织成一个完整的系统
- 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

- 数据流风格：批处理序列、管道—过滤器
- 调用/返回风格：主程序/子程序、面向对象、层次结构
- 独立构建风格：进程通信、事件驱动系统（隐式调用）
- 虚拟机风格：解释器、基于规则的系统
- 仓库风格：数据库系统、超文本系统黑板系统

批处理序列

管道—过滤器

构件为一系列固定顺序的**计算单元**，构件之间只通过数据传递交互，每个处理步骤是一个独立的程序，每一步必须在其前一步结束后才能开始，**数据必须是完整的，以整体的方式传递。**

每个构件都有一组输入和输出，构件读输入的数据流经过内部处理,然后产生输出数据流,这个过程通常是通过  
对输入数据流的变换或计算来完成的,包括通过计算和增加信息以丰富数据、通过浓缩和删除以精简数据，通过  
改变记录方式以转化数据和递增地转化数据等.这里的**构件称为过滤器,连接件就是数据传输的管道**，将一个过滤器的输出传到另一个过滤器的输入。

**早期翻译器就是采用的这种架构.要一步一步处理的，均可考虑采用此架构风格**

主程序/子程序

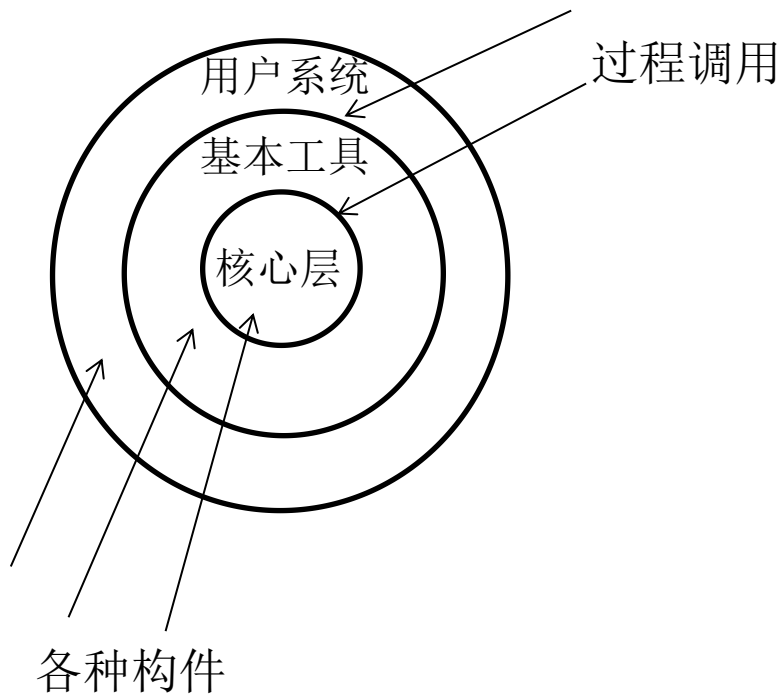
面向对象

层次结构

单线程控制，把问题划分为若干个处理步骤，构件即为主程序和子程序，子程序通常可合成为模板。过程调用作为交互机制，即充当连接件的角色。通用关系具有层次性，其语义逻辑表现为主程序的正确性取决于它调用的子程序的正确性

构件是对象，对象是抽象数据类型的实例。在抽象数据类型中，数据的表示和它们的相应操作被封装起来，对象的行为体现在其接受和请求的动作。连接件即是对象间交互的方式，对象是通过函数和过程的调用来交互的

构件组织成一个层次结构，连接件通过决定层间如何交互的协议来定义。每层为上一层提供服务，使用下一层的服务，只能见到与自己邻接的层。通过层次结构，可以将大的问题分解为若干个渐进的小问题逐步解决，可以隐藏问题的复杂度。修改某一层，最多影响其相邻的两层（通常只能影响上层）



优点：

- 1、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实现。
- 2、不同的层次处于不同的抽象级别：  
越靠近底层，抽象级别越高；  
越靠近顶层，抽象级别越低；
- 3、由于每一层最多只影响两层，同时只要给相邻层提供相同的接口，允许每层用不同的方法实现，同样为软件复用提供了强大的支持

缺点：

- 1、并不是每个系统都可以很容易的划分为分层的模式。
- 2、很难找到一个合适的、正确的层次抽象方法

进程通信

事件驱动系统

(隐式调用)

构件是独立的过程，连接件是消息传递，构件通常是命名过程，消息传递的方式可以是点对点、异步或同步方式，以及远程过程（方法）调用等

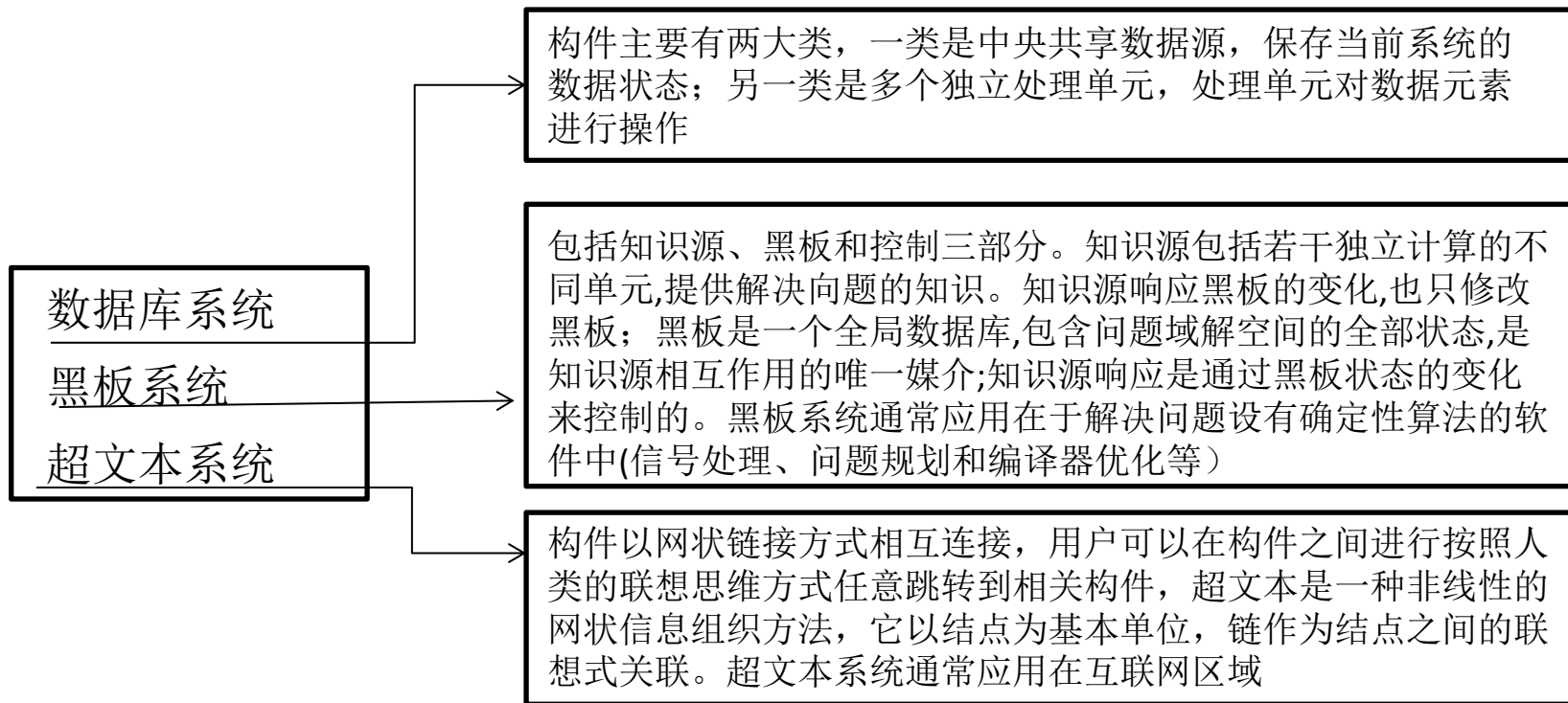
构件不直接调用一个过程，而是触发或广播一个或多个事件。构件中的过程在一个或多个事件中注册，当某个事件被触发时，系统自动调用在这个事件中注册的所有过程。一个事件的触发就导致了另一个模块中的过程调用。这种风格中的构件是匿名的过程，它们之间交互的连接件往往是以过程之间的隐式调用来实现的。主要优点是软件复用提供了强大的支持，为构件的维护和演化带来了方便；其缺点是构件放弃对系统计算的控制

解释器

基于规则的系统

解释器通常包括一个完成解释工作的解释引擎，一个包含将被解释的代码的存储区，一个记录解释引擎当前工作状态的数据结构，以及一个记录源代码被解释执行的进度的数据结构，具有解释器风格的软件中含有一个虚拟机，可以仿真硬件的执行过程和一些关键应用，其缺点是执行效率比较低

基于规则的系统包括规则集、规则解释器、规则/数据选择器和工作内存，一般用在人工智能领域和DSS中



现代集成编译环境一般采用这种架构风格



DESIGNER:王川林  
软件架构设计



# THANK YOU